

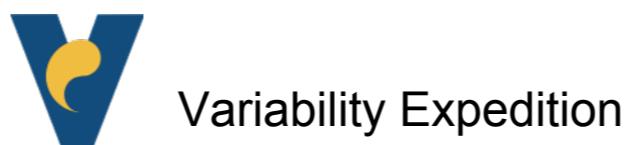


Algorithms

Tianmu Li
litianmu1995@ucla.edu

This is a modified version of slides by Dr. John Lee,
Dr. Rani Ghaida,
Dr. Abde Ali Kagalwalla, Dr. Yasmine Badr and
Saptadeep Pal.

A lot of the material is based on
Introduction to Algorithms by Cormen et al.



What is an Algorithm?

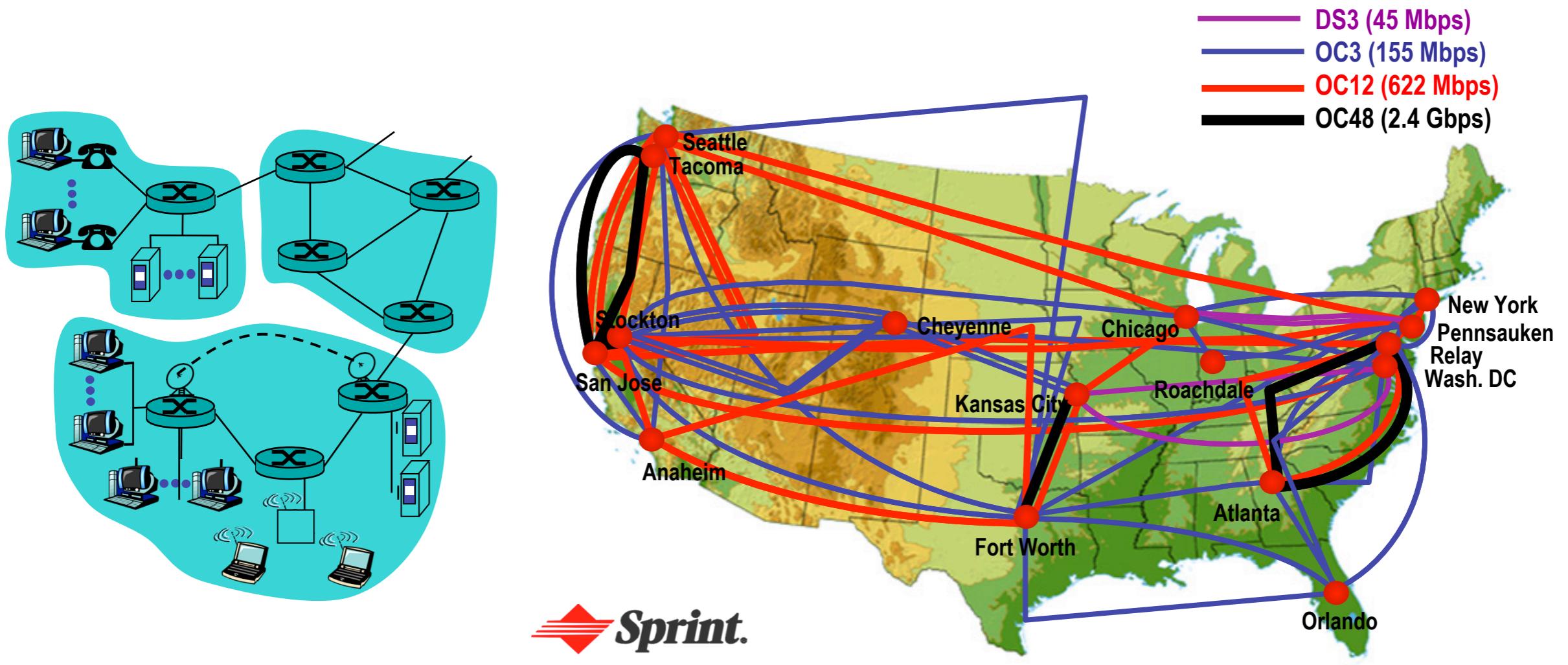
- An algorithm is a well-defined computational procedure that takes an input and produces some output.
It is thus a set of computational steps that transform the input into the output.



Agenda

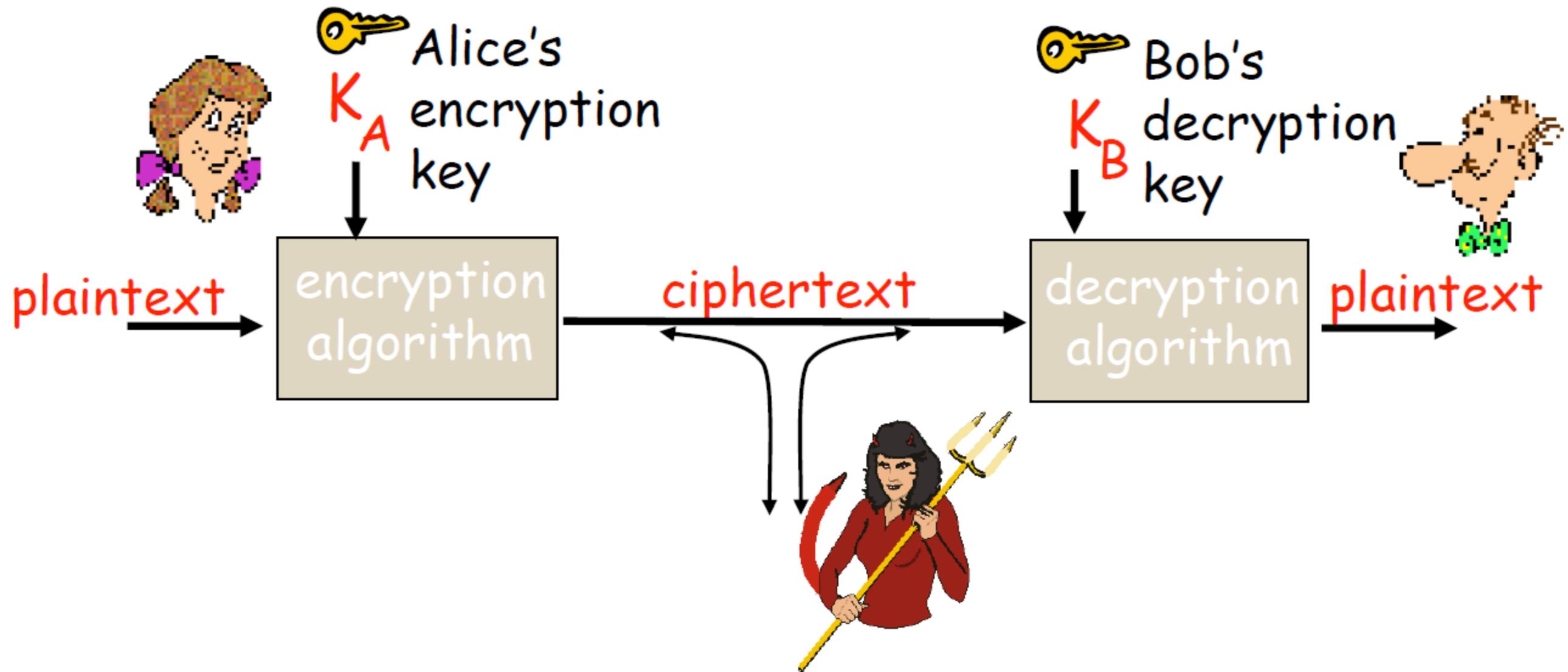
- What sort of problems are solved with algorithms?
- Examples of Algorithms
 - Search Algorithms
 - Gale Shapley Stable Marriage Algorithm
 - Shortest-Path Problem
- Efficiency of Algorithms
 - Sorting Algorithms
 - *Exercise*: Bubble Sort
- Algorithmic Techniques
 - Greedy Algorithm
 - Example: Activity Selection Problem
- *Mini-project*: Traveling Salesman Problem
- Research Topics

What sort of problems are solved with algorithms?



- Internet - Manage and manipulate data
 - E.g., (1) search engines, (2) finding good routes on which data will travel

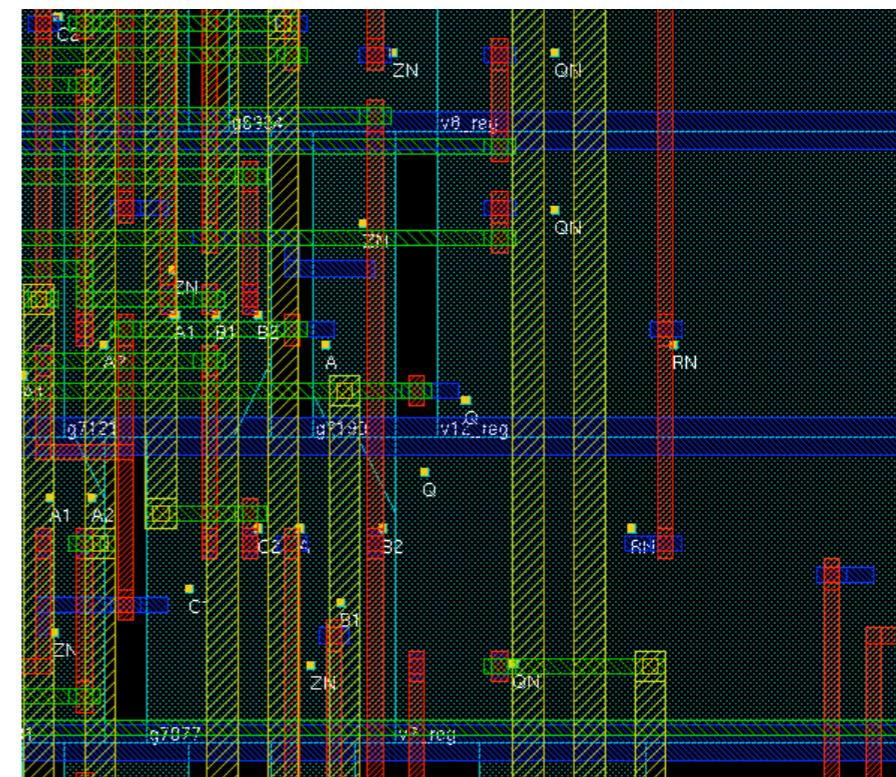
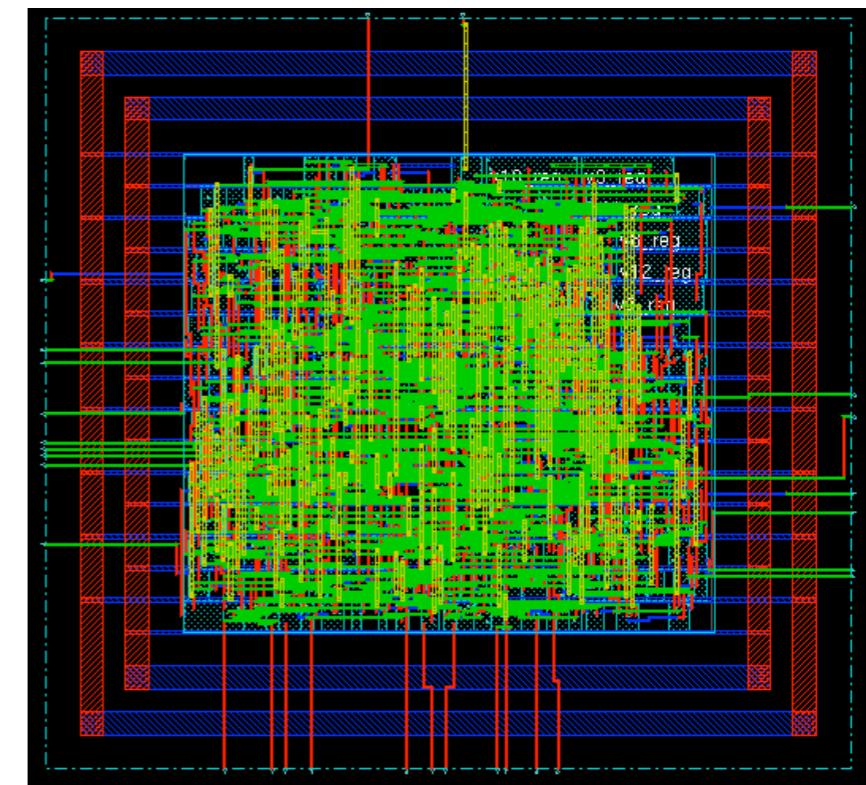
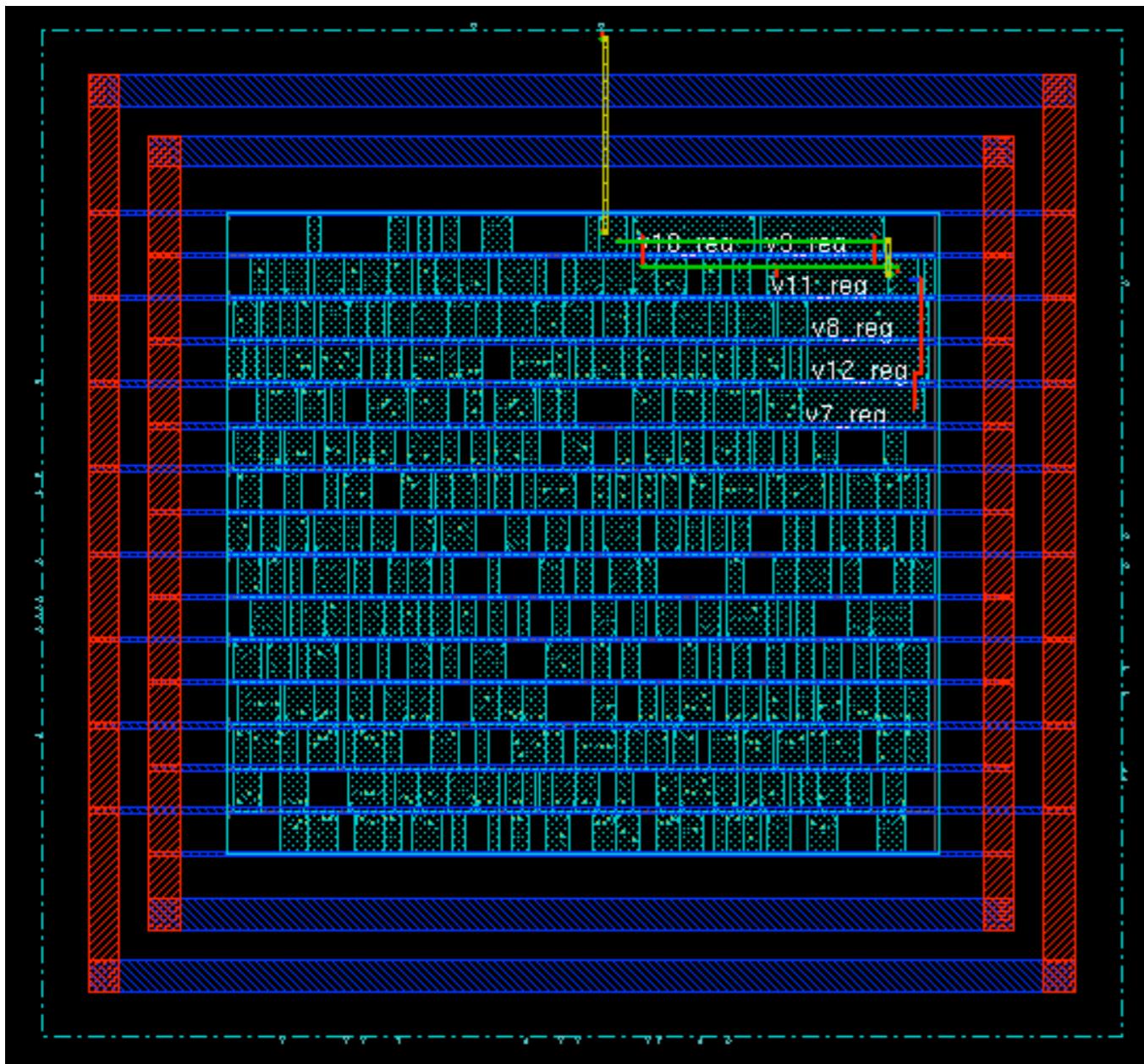
What sort of problems are solved with algorithms?



- **Data security – cryptography**
 - Encryption algorithm → computational steps to transform plaintext to ciphertext
 - Decryption algorithm → computational steps to transform ciphertext back to plaintext

What sort of problems are solved with algorithms?

- **NanoElectronics – Design Automation**
 - E.g., components placement and routing



What sort of problems are solved with algorithms?

- **Scheduling problems**
 - E.g., Flight scheduling
 - Crew preferences
 - Airplane types
 - Weather
 - Fuel Costs
 - Delays

• LH 3666	Vancouver	13:05	13:25
• LH 4012	Genf	13:10	
• LH 4576	Turin	13:10	
• LH 418	Bruessel	13:10	
• LH 1172	Washington	13:10	13:25
• LH 648	Paderborn Lippstadt	13:15	
• LH 470	Almaty	13:20	
• LH 180	Toronto	13:20	
• LH 4376	Berlin-Tegel	13:20	
	Toulouse	13:20	13:50
	Istanbul	13:20	13:55
	Couille		

From <http://www.worldofstock.com/slides/TRA2599.jpg>

What sort of problems are solved with algorithms?

- **Artificial Intelligence**

- E.g., IBM Watson supercomputer beats all-time best Jeopardy! players on Feb 2011

Daily Double betting



Final Jeopardy betting

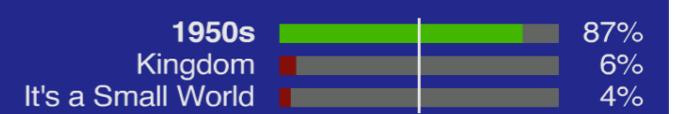


Clue selection



Confidence Threshold for Buzz-in

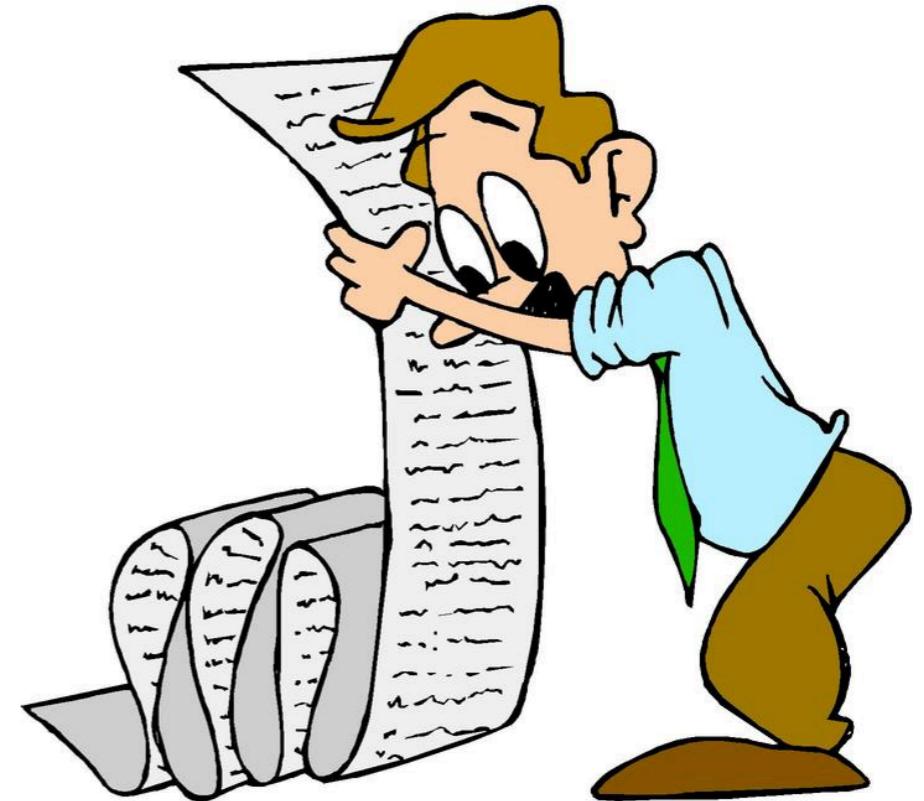
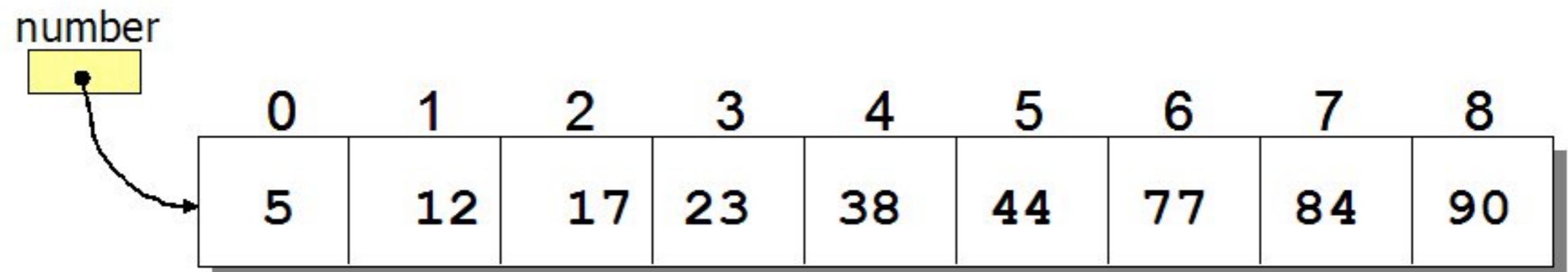
Watson



Search Algorithms

- How to search for an element in a list?
 1. **Sequential search:** Go through each element and check if it is the required element
 - Best case?
 - Worst case?
 2. What if the list is sorted?

Can we do better?



Source: <http://www.spinachandyoga.com/how-to-deal-with-a-long-to-do-list-with-zen/>

Search Algorithms

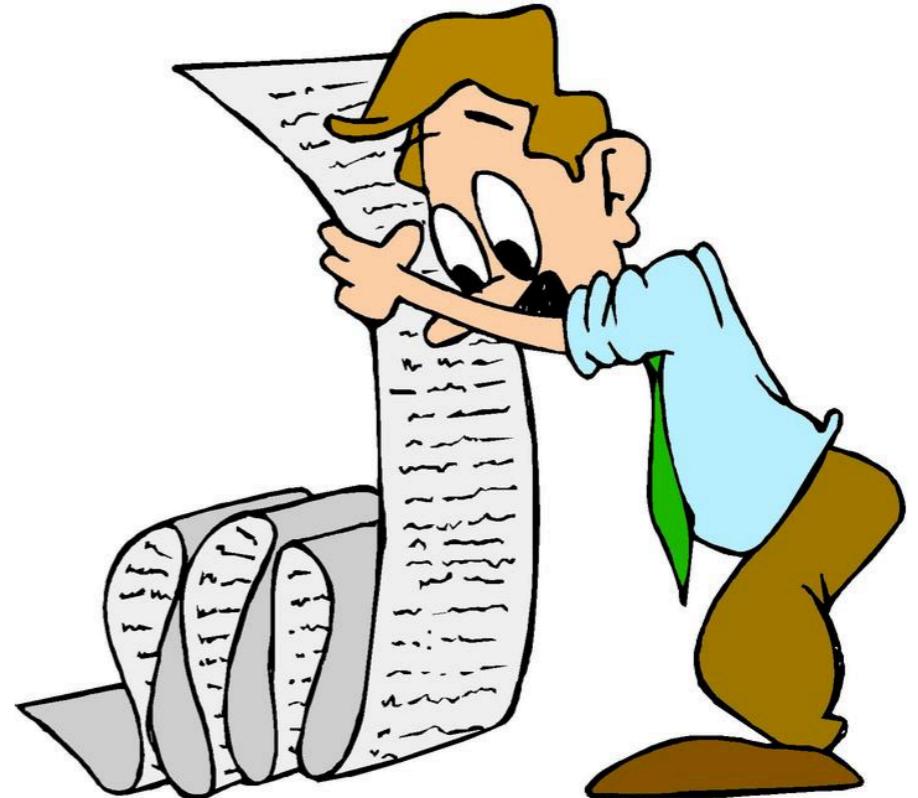
- How to search for an element x in a list?

1. **Sequential search:** Go through each element and check if it is the required element
 - Best case?
 - Worst case?
2. What if the list is sorted?

Can we do better?

YES

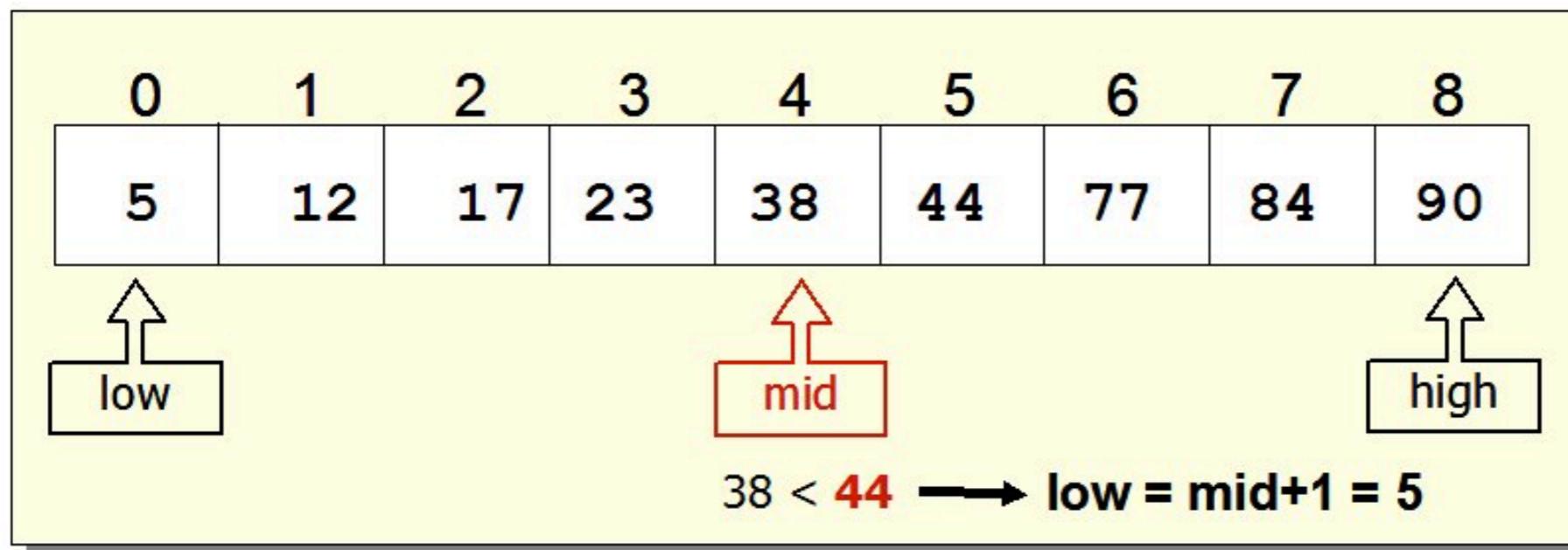
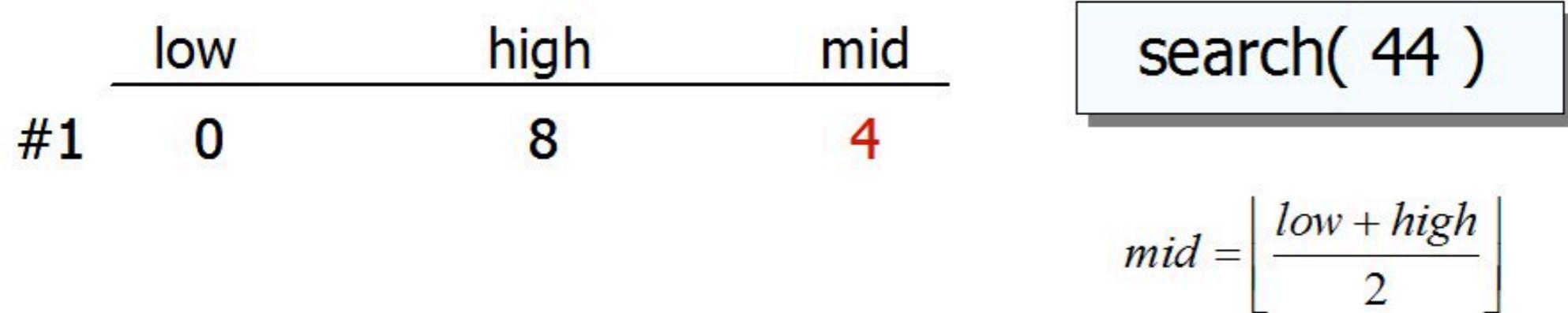
BINARY SEARCH



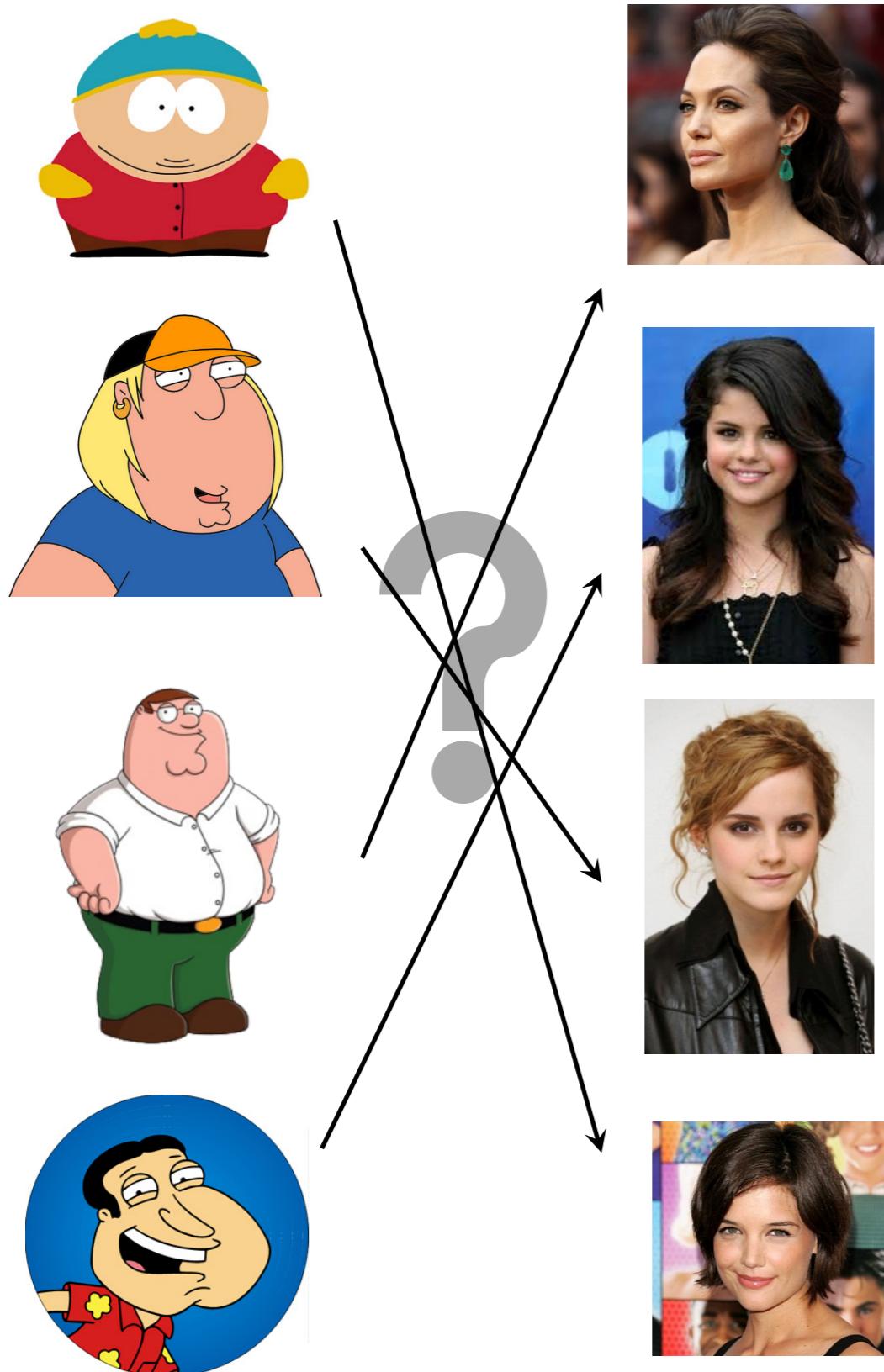
Source: <http://www.spinachandyoga.com/how-to-deal-with-a-long-to-do-list-with-zen/>

Search Algorithms: Binary Search

- If list is **sorted**:
 - Split list into two halves. If x is greater than element in the middle then discard the left half. Otherwise discard the right half. Repeat until you find x .



Example: Stable Marriage Problem



Input:
Men, Women, Preferences

Algorithm

Output:
1- All married
2- Stable marriages (i.e., no one prefers someone else who also prefers them)

Example: Stable Marriage Problem

- Algorithm to solve this?

		Preferences							
Proposors	Acceptors	Acceptor Table				Proposor Table			
1	1	1	3	2	4	1	2	3	4
2	2	2	4	1	2	2	4	1	3
3	3	3	2	3	1	3	1	2	4
4	4	4	3	1	4	4	2	3	1

Gale Shapley Stable Marriage Algorithm

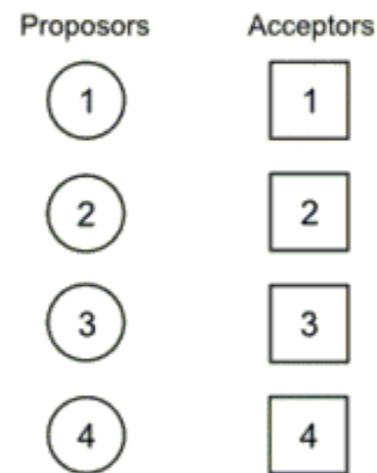
1. In each round, unengaged man "proposes" to the most-preferred woman to whom he has not yet proposed
2. Each woman then considers all her suitors and tells the one she most prefers "Maybe" and all the rest of them "No".
3. She is then provisionally "engaged" to the suitor she most prefers so far, and that suitor is likewise provisionally engaged to her

Pseudocode

```
function stableMatching {
    Initialize all  $m \in M$  and  $w \in W$  to free
    while  $\exists$  free man  $m$  who still has a woman  $w$  to propose to {
         $w = m$ 's highest ranked such woman to whom he has not yet proposed
        if  $w$  is free
             $(m, w)$  become engaged
        else some pair  $(m', w)$  already exists
            if  $w$  prefers  $m$  to  $m'$ 
                 $(m, w)$  become engaged
                 $m'$  becomes free
            else
                 $(m', w)$  remain engaged
    }
}
```

Gale Shapley Stable Marriage Algorithm

Round : 1



Proposal pool	
1	3
2	1 4
3	
4	2

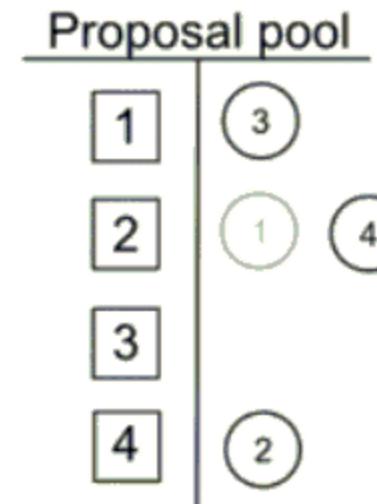
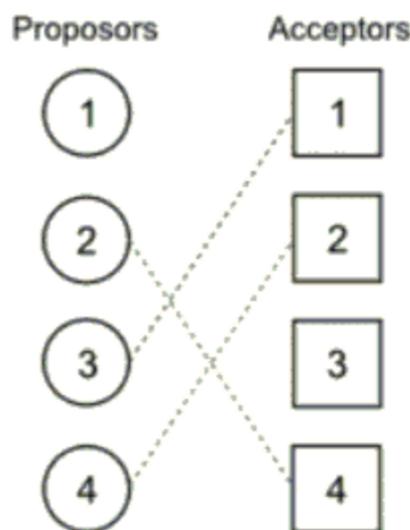
- 1-4 propose, as none are currently tentatively attached

Preferences

	□ → ○				○ → □			
	Acceptor Table				Proposor Table			
1	1	3	2	4	(1)	2	1	3
2	3	4	1	2	(2)	4	1	2
3	4	2	3	1	(3)	1	3	2
4	3	2	1	4	(4)	2	3	1

Gale Shapley Stable Marriage Algorithm

Round : 1



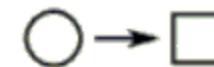
- 1 accepts 3's proposal – no better offer.
- 2 accepts 4's proposal as 4 is more preferable to 1.
- 3 receives no offer.
- 4 accepts 2's proposal – no better offer.

Preferences



Acceptor Table

1	1	3	2	4
2	3	4	1	2
3	4	2	3	1
4	3	2	1	4

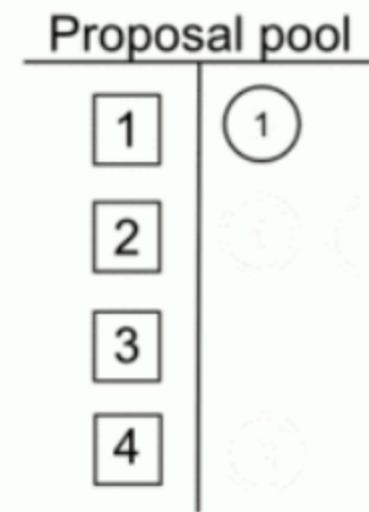
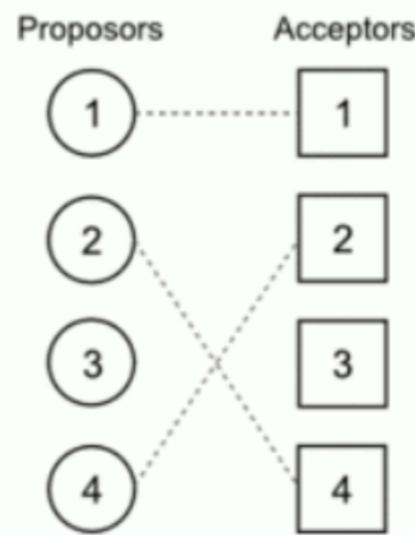


Proposor Table

1	2	1	3	4
2	4	1	2	3
3	1	3	2	4
4	2	3	1	4

Gale Shapley Stable Marriage Algorithm

Round : 2



- 1 drops 3's proposal in favour of 1 as this is higher in its preference table. 3 returns to the proposal pool.

Preferences

□ → ○

Acceptor Table

1	1	3	2	4
2	3	4	1	2
3	4	2	3	1
4	3	2	1	4

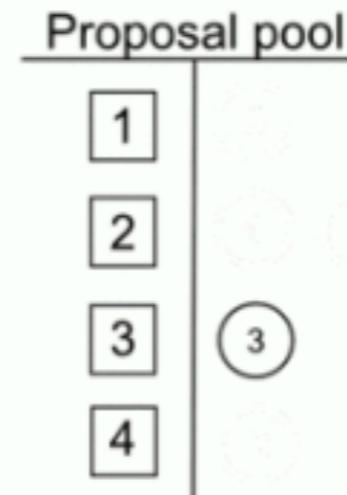
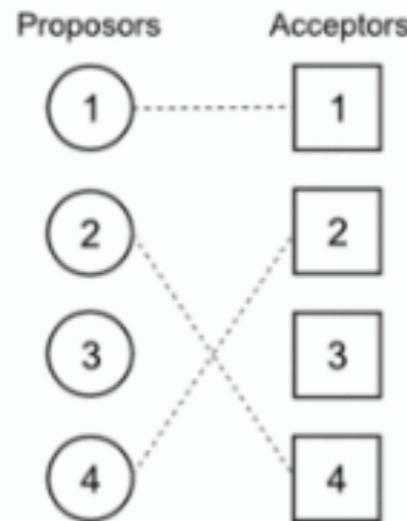
○ → □

Proposor Table

1	2	1	3	4
2	4	1	2	3
3	1	3	2	4
4	2	3	1	4

Gale Shapley Stable Marriage Algorithm

Round : 3



- 3 proposes to 3

Preferences

□ → ○

Acceptor Table

1	1	3	2	4
2	3	4	1	2
3	4	2	3	1
4	3	2	1	4

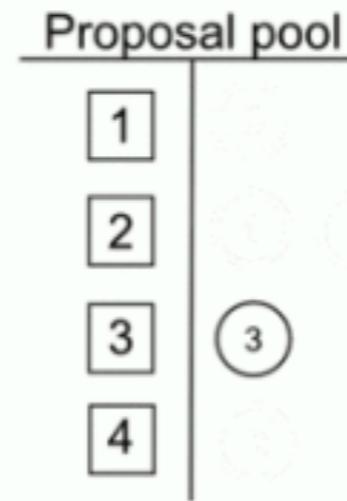
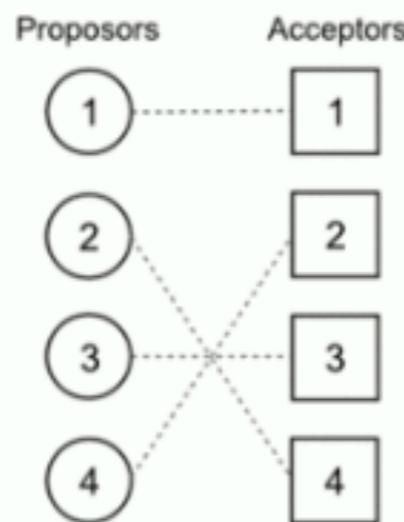
○ → □

Proposor Table

1	2	1	3	4
2	4	1	2	3
3	1	3	2	4
4	2	3	1	4

Gale Shapley Stable Marriage Algorithm

Round : 3



- 3 accepts 3, not having a better offer

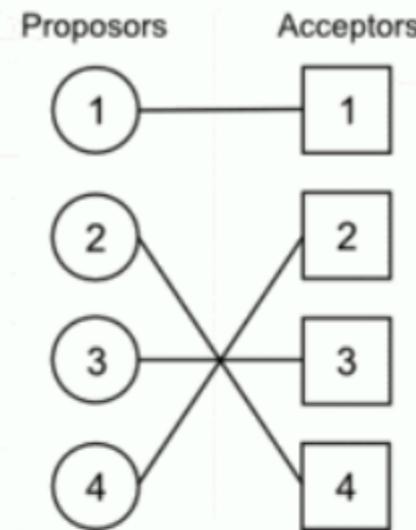
Preferences

$\square \rightarrow \circlearrowleft$ Acceptor Table $\circlearrowright \rightarrow \square$ Proposor Table

	1	3	2	4		1	2	1	3	4
1	1	3	2	4		1	2	1	3	4
2	3	4	1	2		2	4	1	2	3
3	4	2	3	1		3	1	3	2	4
4	3	2	1	4		4	2	3	1	4

Gale Shapley Stable Marriage Algorithm

Finish

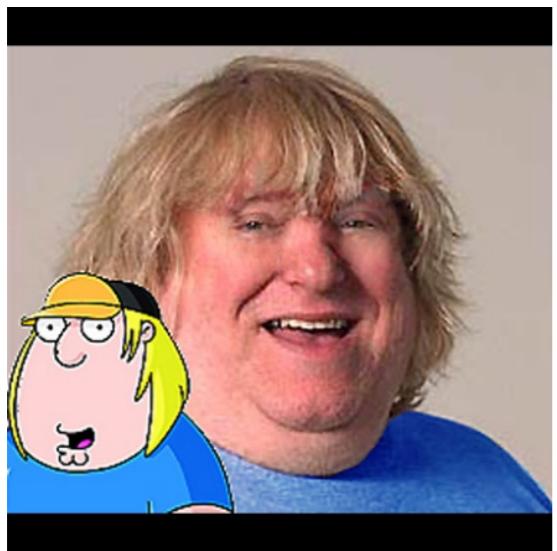


- No two members {P,A} would prefer one-another over their current pairing

Preferences

		Acceptor Table				Proposor Table			
		□ → ○	Acceptor Table			○ → □	Proposor Table		
			1	2	3		1	2	3
1	2		1	3	2	4	1	2	3
2	3		3	4	1	2	2	4	1
3	4		4	2	3	1	3	1	3
4	1		3	2	1	4	4	2	3

Our Happy Couples!



Shortest-Path Problem

- Google Maps - determining the fastest route

Driving directions to Diddy Riese

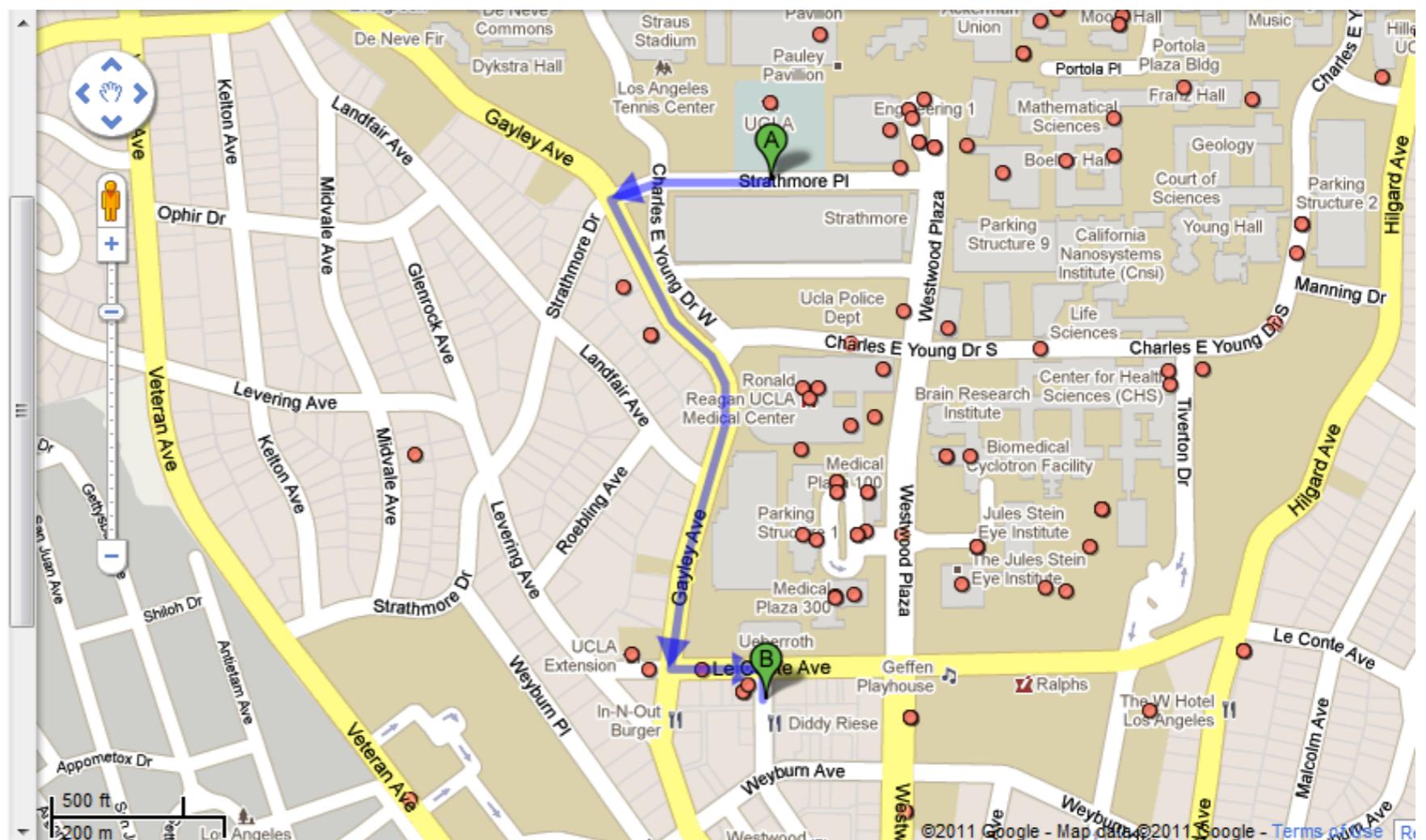
Suggested routes

1. Strathmore PI and Gayley Ave	3 mins
0.6 mi	
2. Strathmore PI and Westwood Plaza	3 mins
0.6 mi	
3. Strathmore PI	3 mins
0.6 mi	

A University of California Los Angeles
405 Hilgard Ave
Los Angeles, CA 90095

1. Head west on Strathmore PI toward Charles E Young Dr
2. Turn left at Gayley Ave
3. Turn left at Le Conte Ave
4. Take the 1st right onto Broxton Ave

B Diddy Riese
926 Broxton Avenue



Shortest-Path Problem

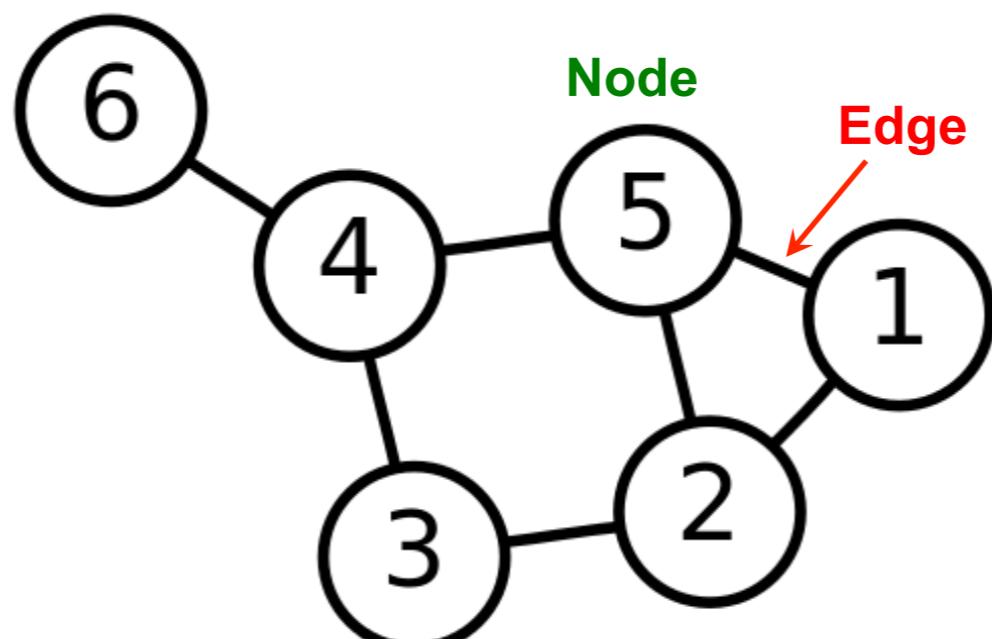
- Google Maps - determining the fastest route



Graph [In Brief]

- Set of :
 - **Nodes/vertices**
 - **edges** connecting the nodes
- Edges can be
 - Undirected: no distinction between the 2 nodes
 - Directed: has a source and a destination

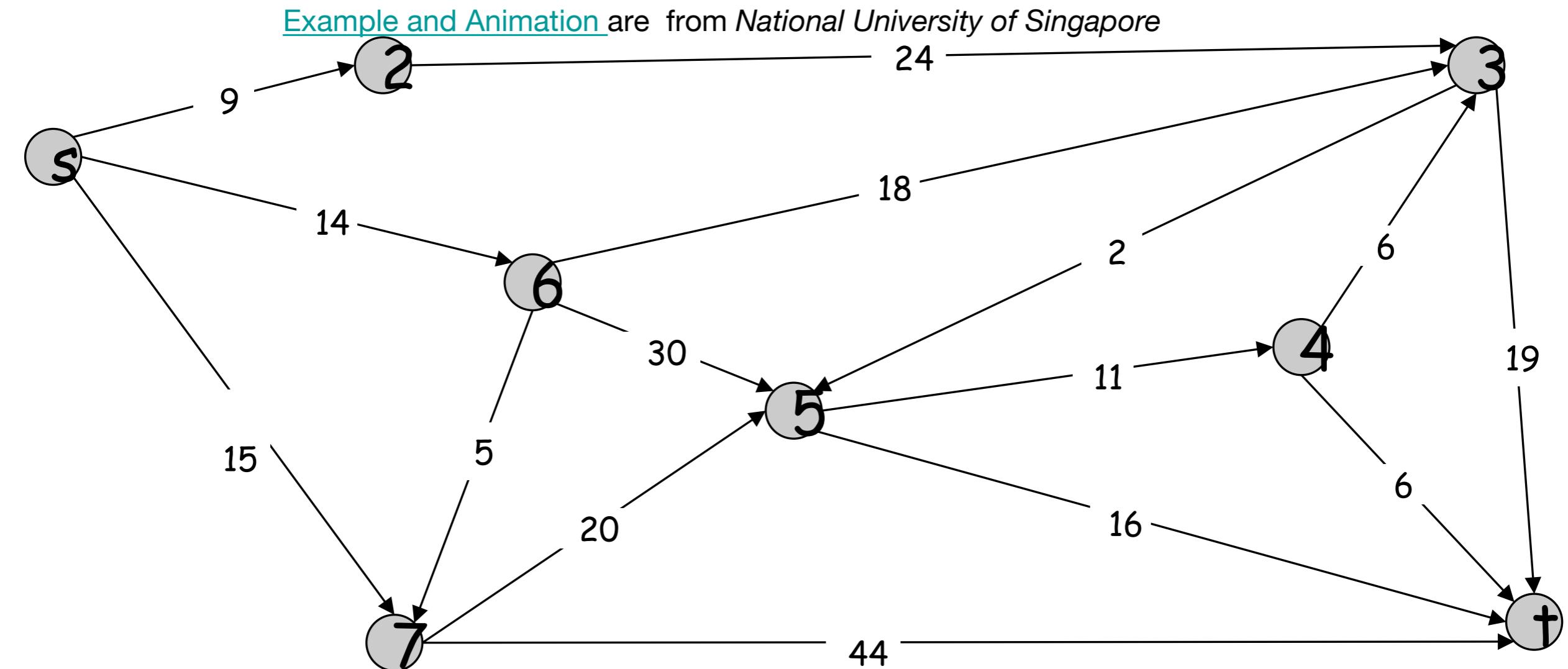
**Undirected
graph**



Source: https://en.wikipedia.org/wiki/Graph_theory

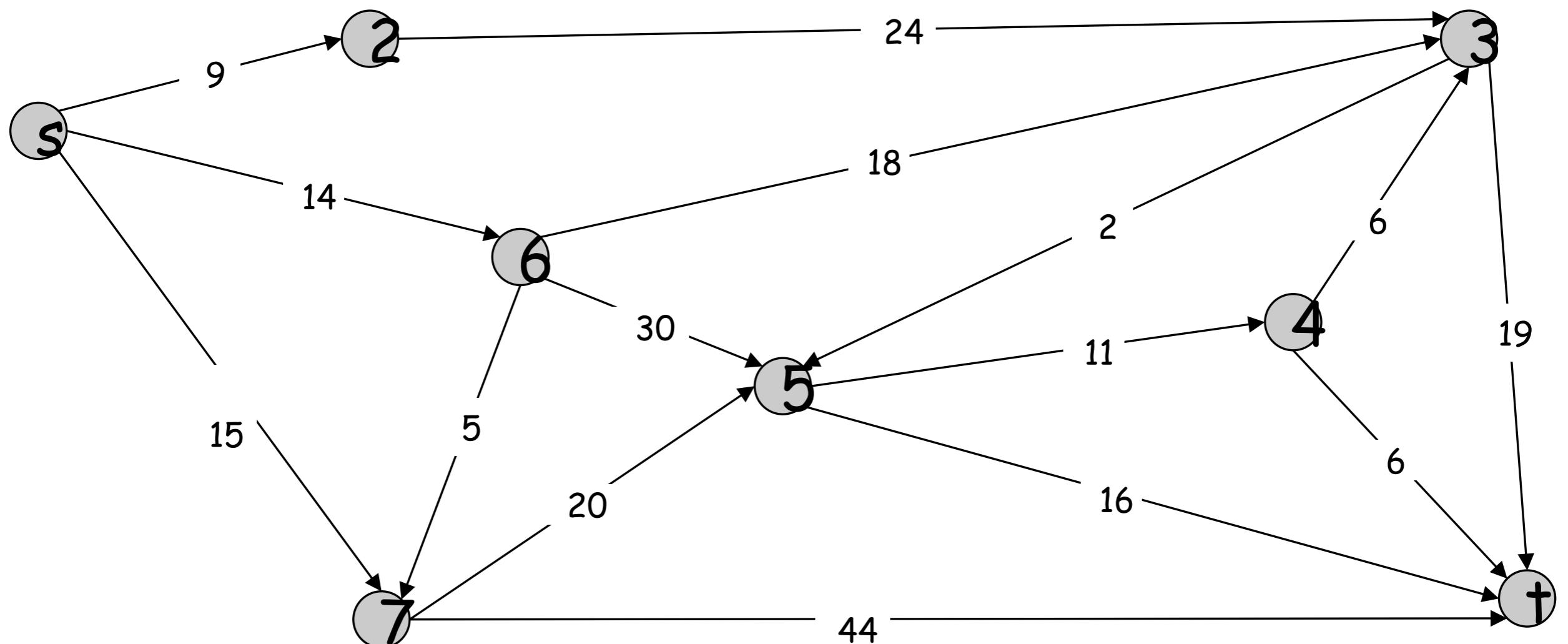
Class Activity: Shortest Path Algorithm

- Can you come up with an algorithm that solves the shortest path problem?
- Work in groups
- What is the shortest path from **s** to **t**?



Dijkstra's Shortest Path Algorithm

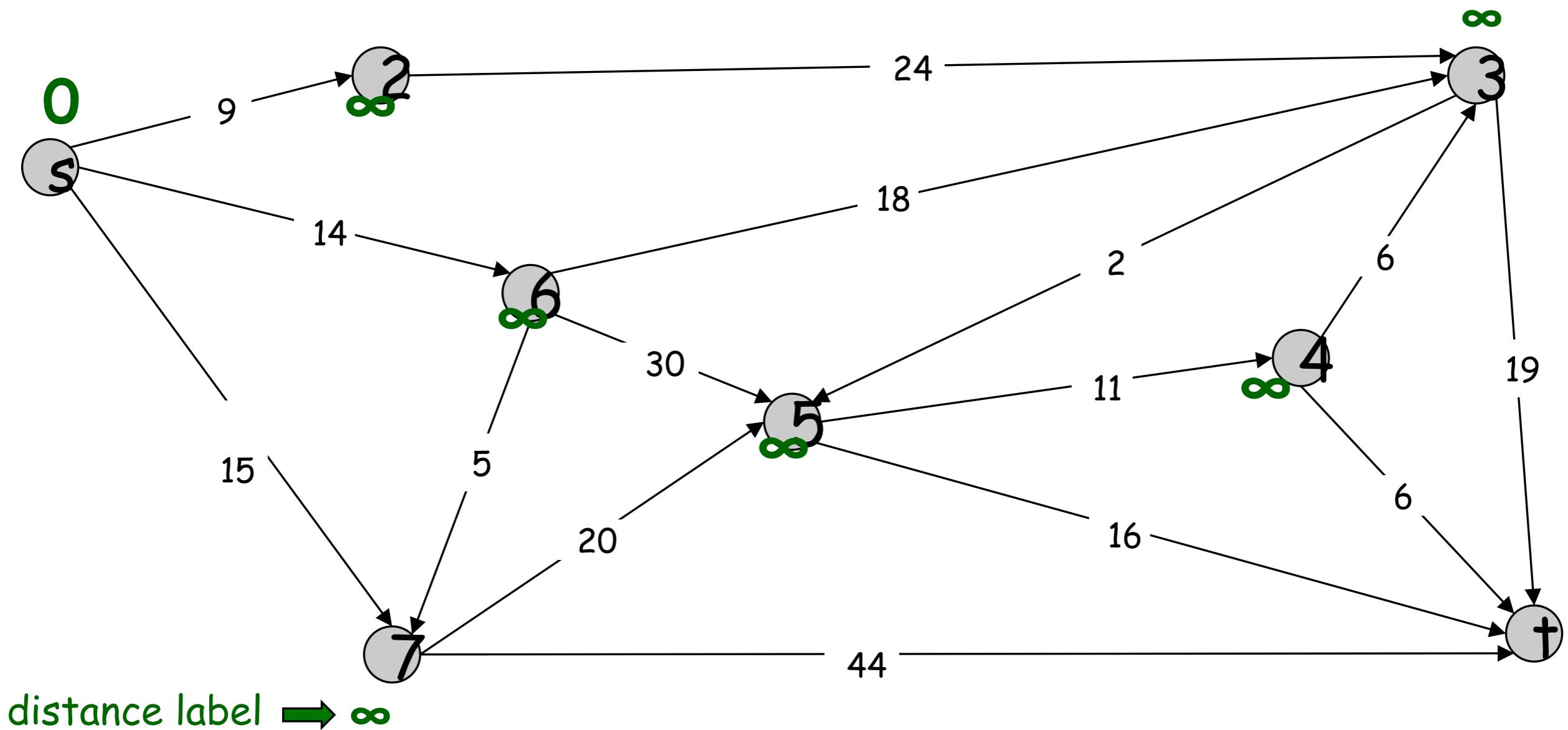
Find shortest path from s to t.



[Example and Animation](#) are from National University of Singapore

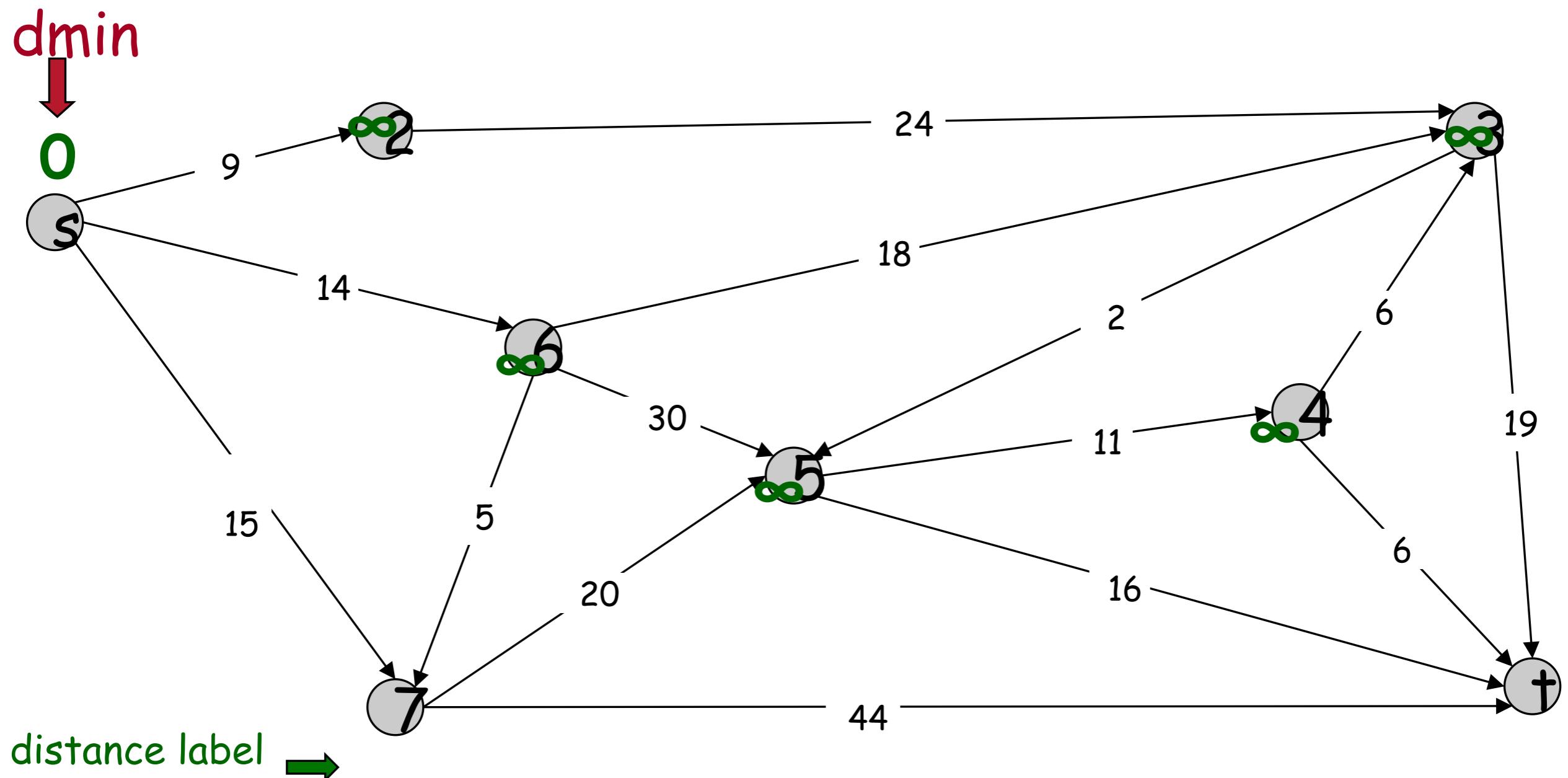
Dijkstra's Shortest Path Algorithm

```
S = { }  
PQ = { s, 2, 3, 4, 5, 6, 7, + }  
B = { }
```



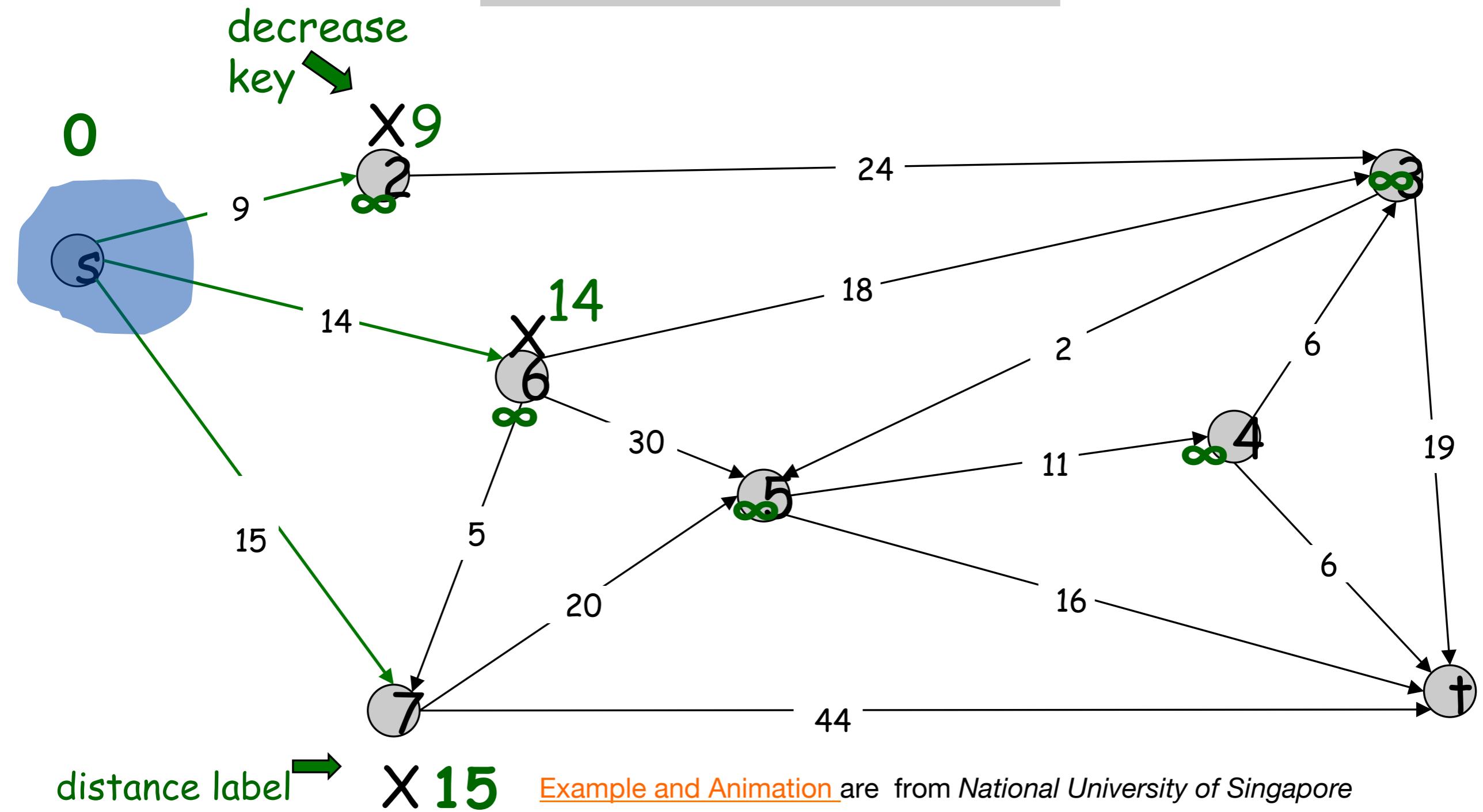
Dijkstra's Shortest Path Algorithm

```
S = { }  
PQ = { s, 2, 3, 4, 5, 6, 7, + }  
B = { }
```



Dijkstra's Shortest Path Algorithm

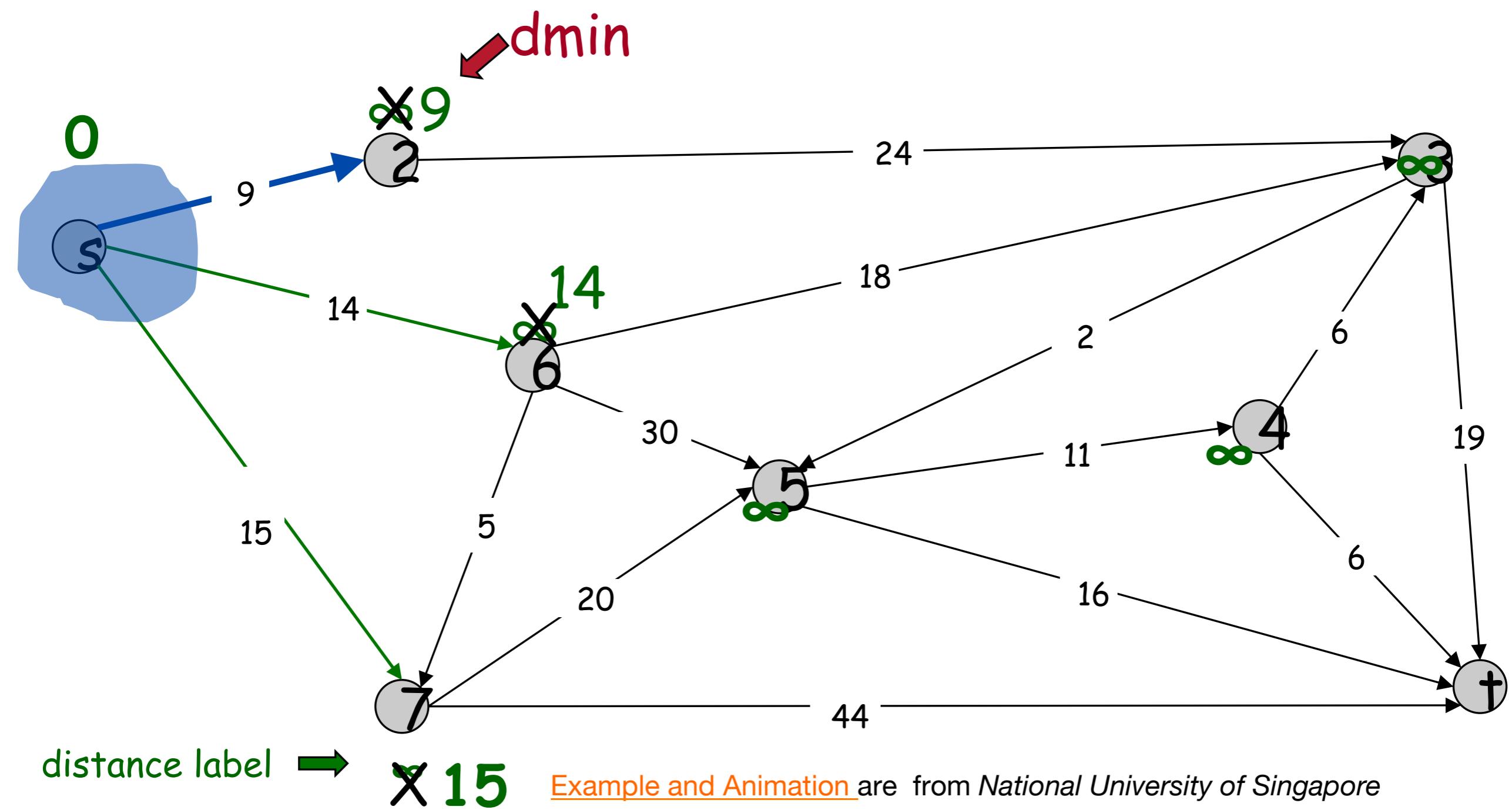
```
S = { s }
PQ = { 2, 3, 4, 5, 6, 7, † }
B = { s }
```



[Example and Animation](#) are from National University of Singapore

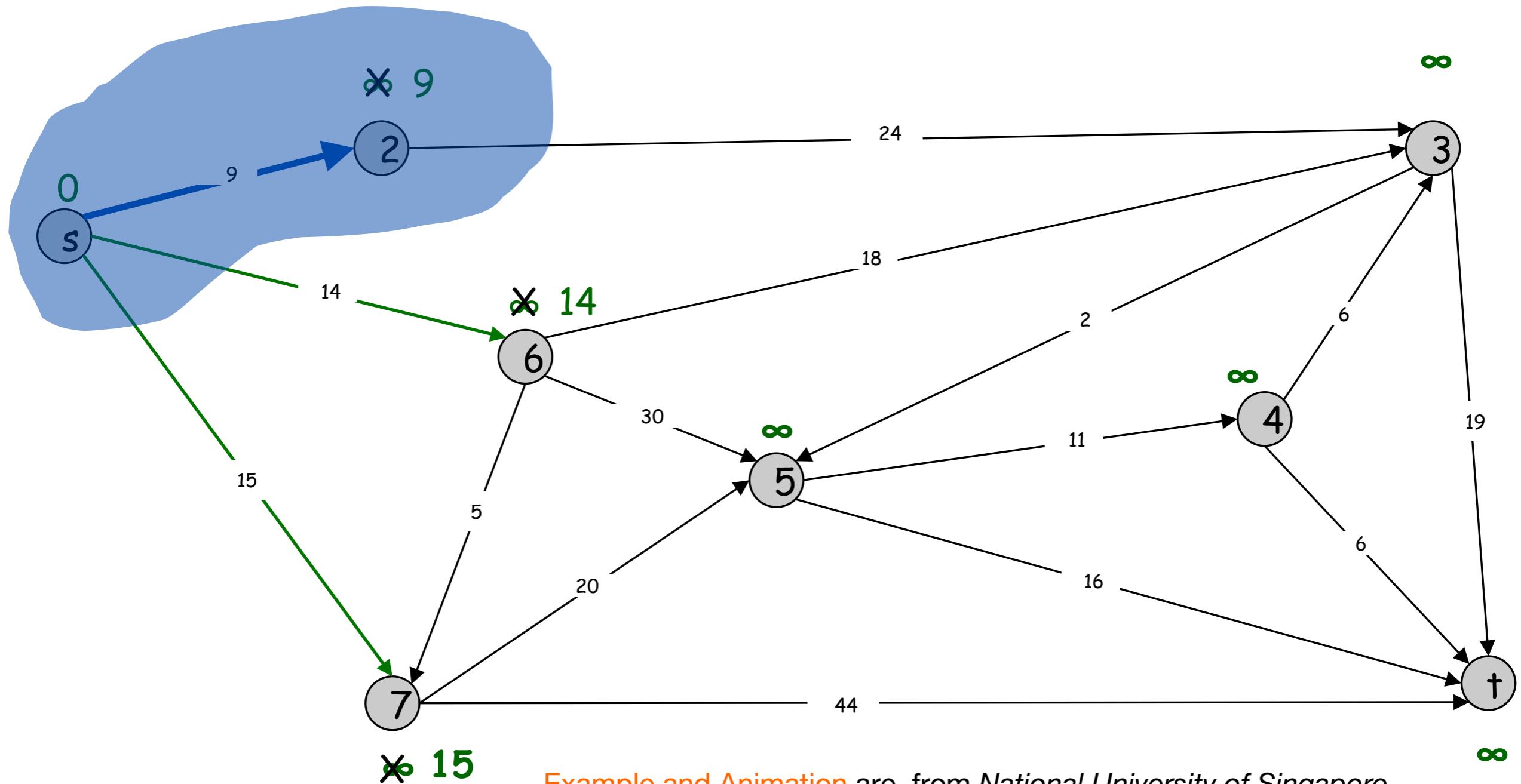
Dijkstra's Shortest Path Algorithm

```
S = { s }
PQ = { 2, 3, 4, 5, 6, 7, + }
B = { s }
```



Dijkstra's Shortest Path Algorithm

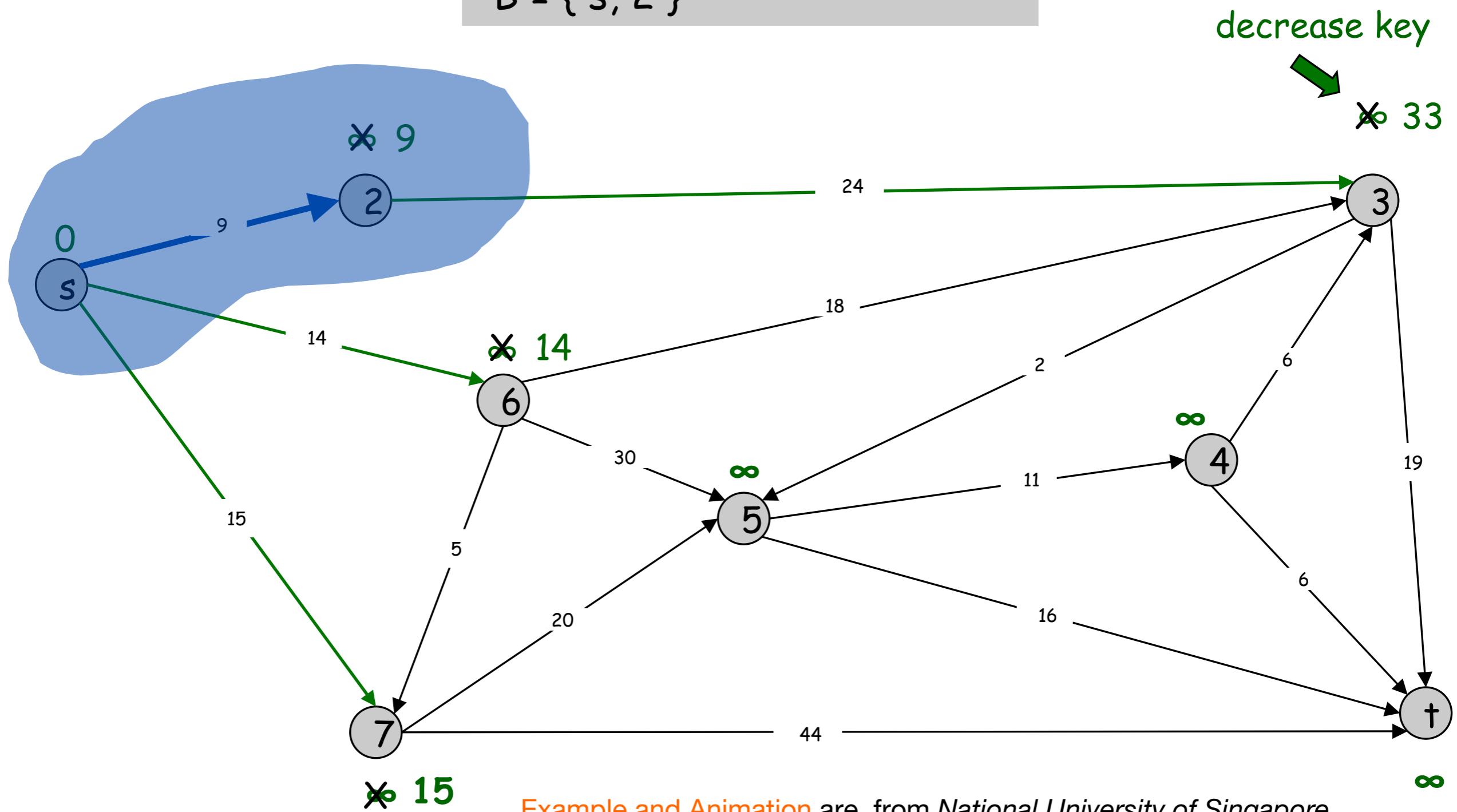
```
S = { s, 2 }
PQ = { 3, 4, 5, 6, 7, † }
B = { s, 2 }
```



[Example and Animation](#) are from National University of Singapore

Dijkstra's Shortest Path Algorithm

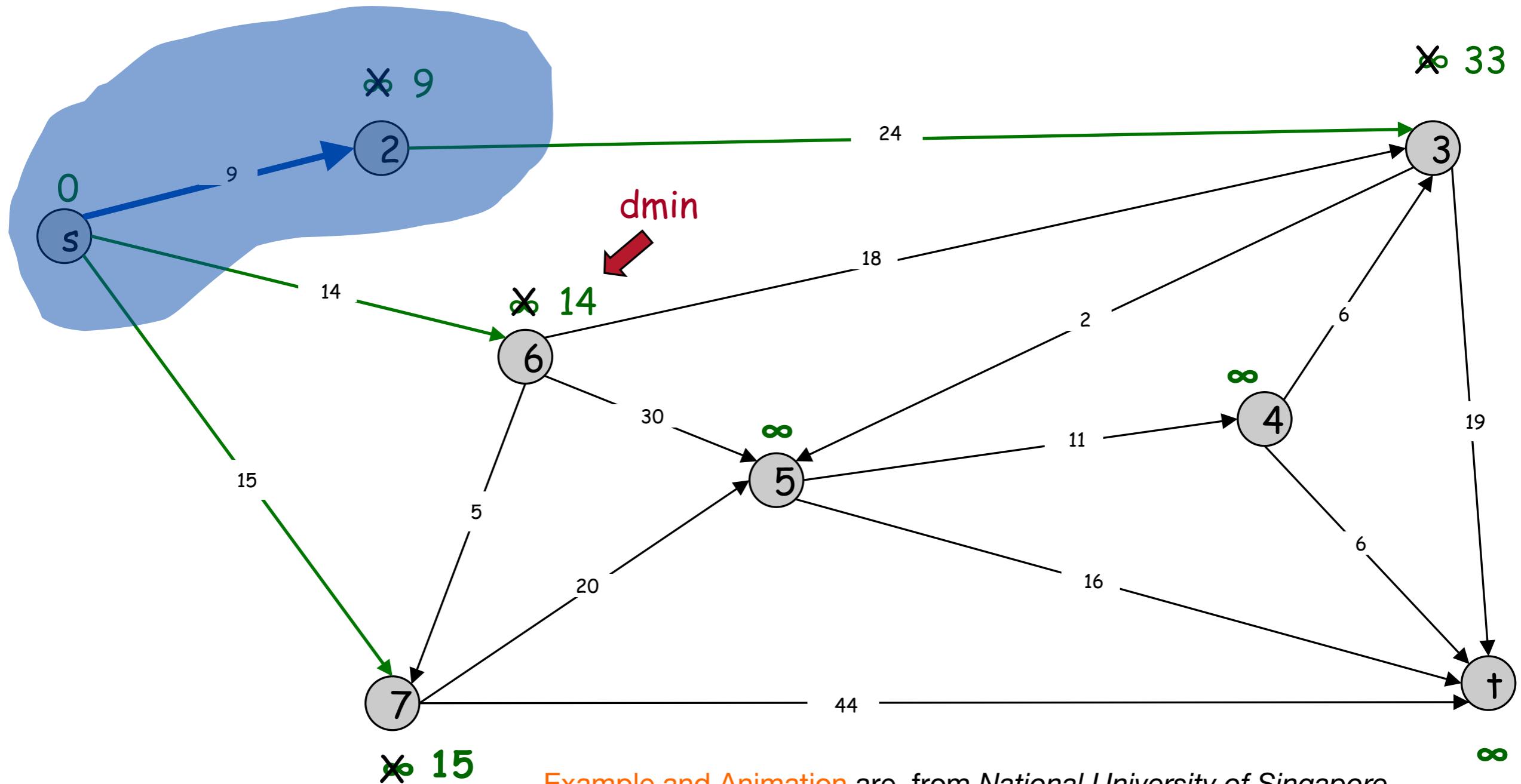
```
S = { s, 2 }  
PQ = { 3, 4, 5, 6, 7, † }  
B = { s, 2 }
```



[Example and Animation](#) are from National University of Singapore

Dijkstra's Shortest Path Algorithm

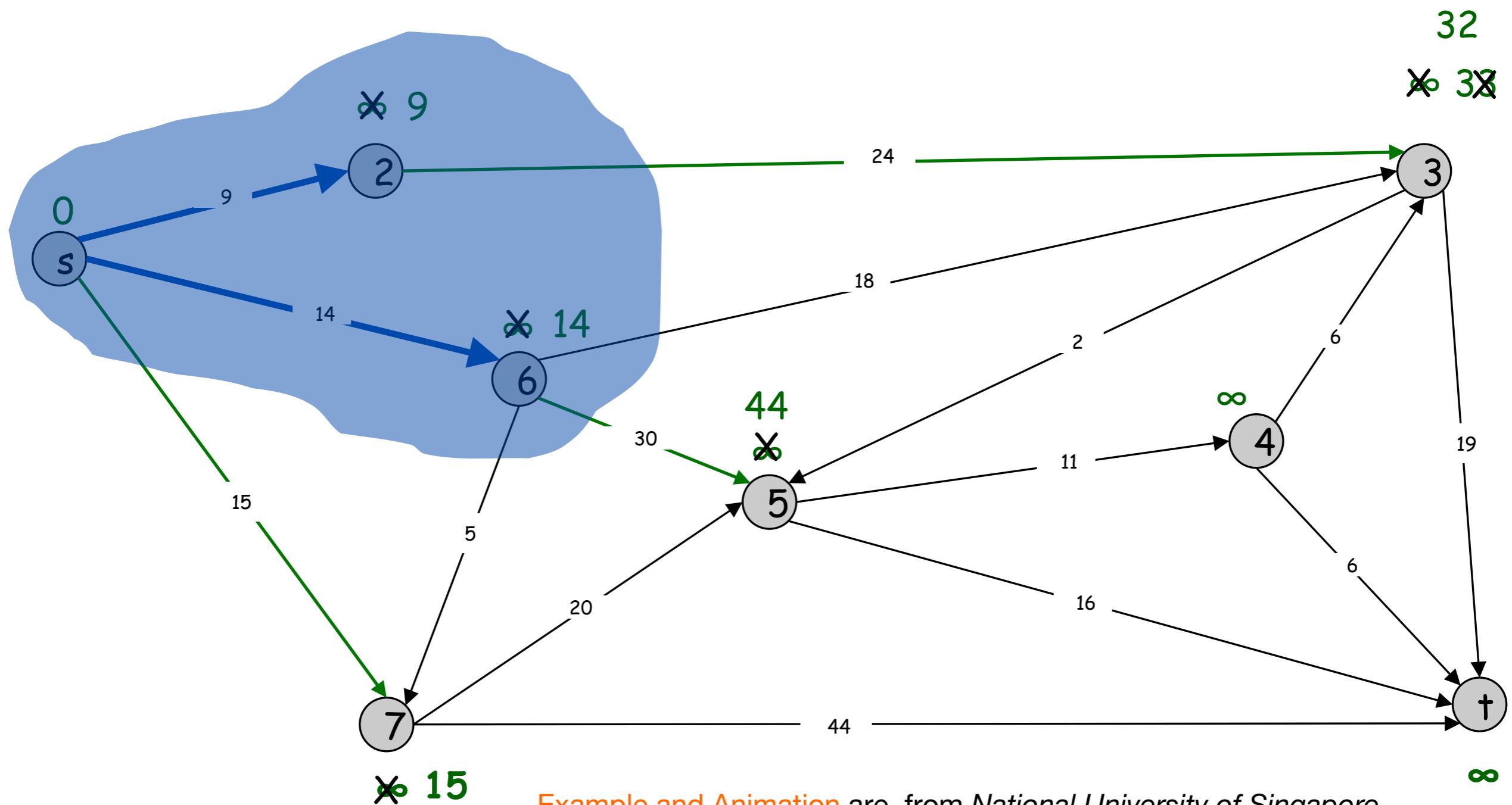
```
S = { s, 2 }
PQ = { 3, 4, 5, 6, 7, † }
B = { s, 2 }
```



[Example and Animation](#) are from National University of Singapore

Dijkstra's Shortest Path Algorithm

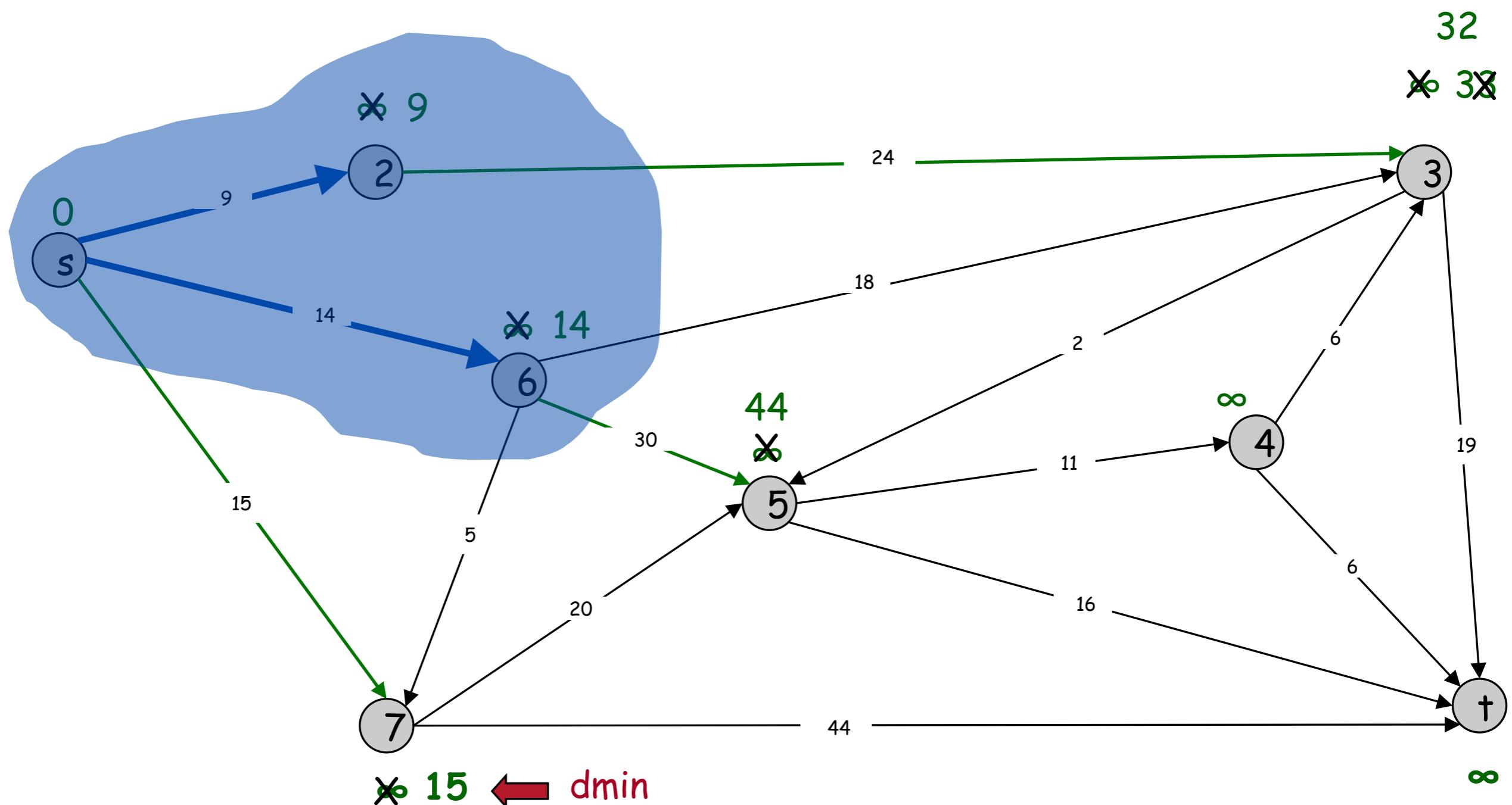
```
S = { s, 2, 6 }
PQ = { 3, 4, 5, 7, + }
B = { s, 6 }
```



[Example and Animation](#) are from National University of Singapore

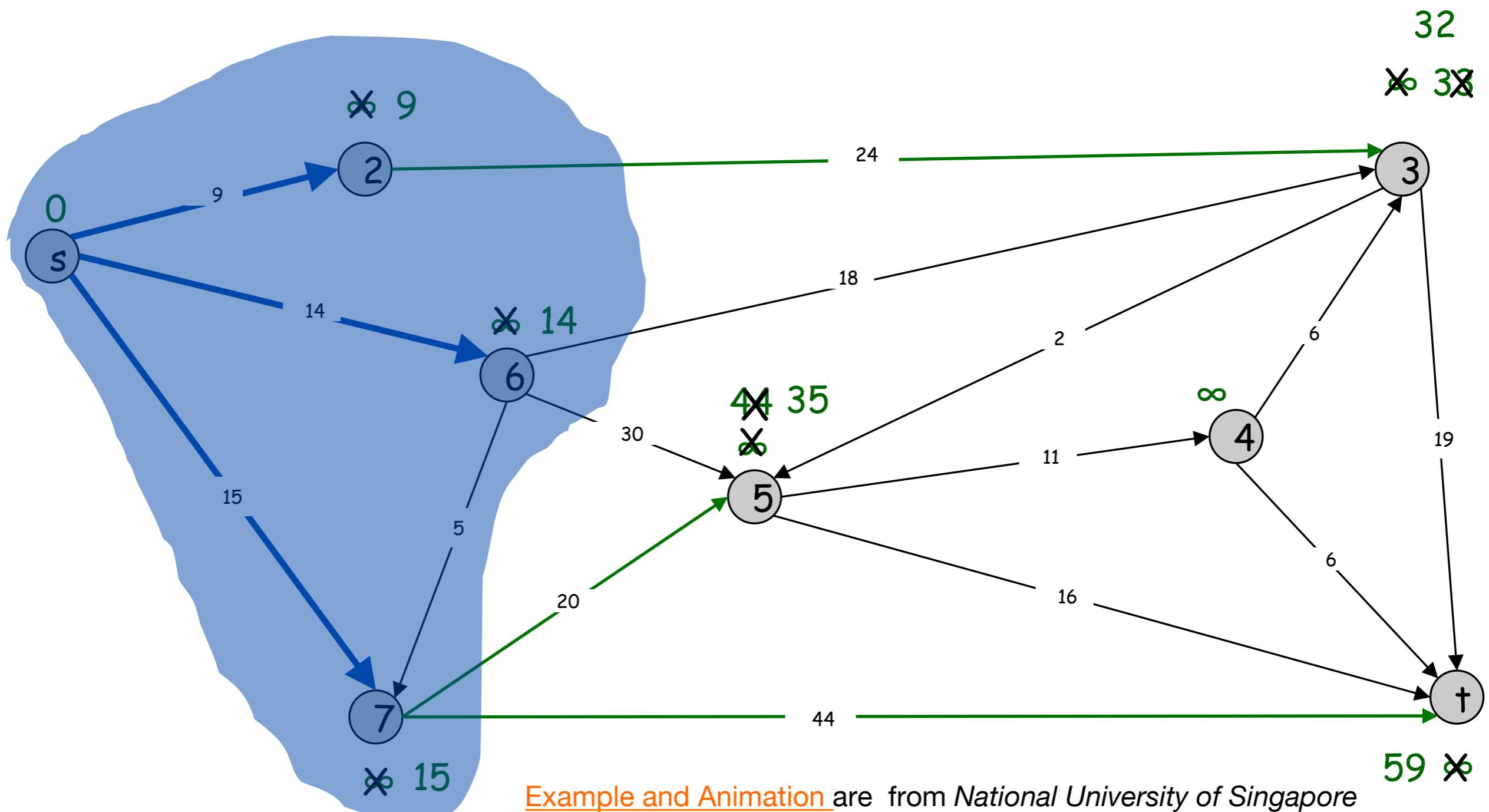
Dijkstra's Shortest Path Algorithm

```
S = { s, 2, 6 }
PQ = { 3, 4, 5, 7, + }
B = { s, 6 }
```



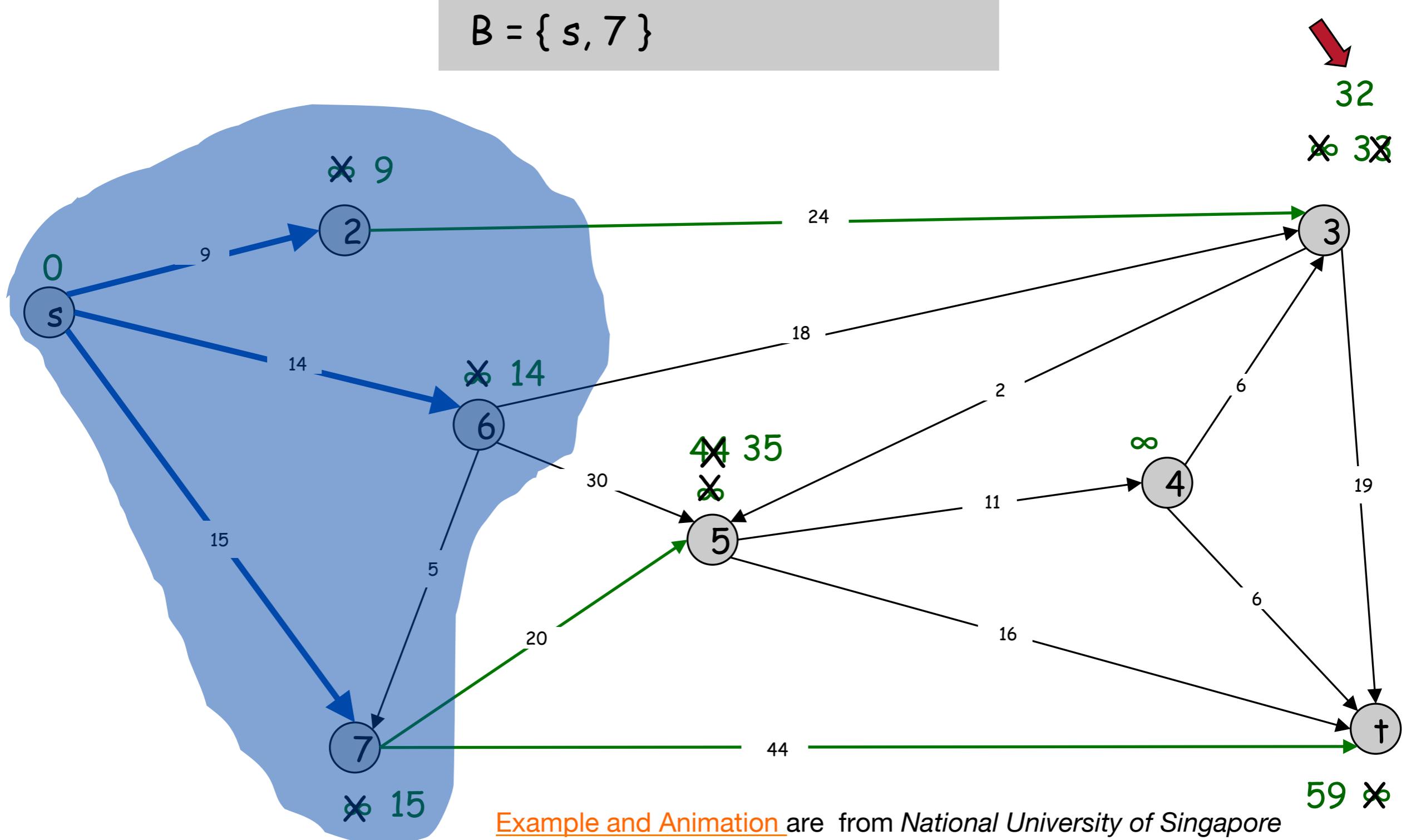
Dijkstra's Shortest Path Algorithm

```
S = { s, 2, 6, 7 }  
PQ = { 3, 4, 5, + }  
B = { s, 7 }
```



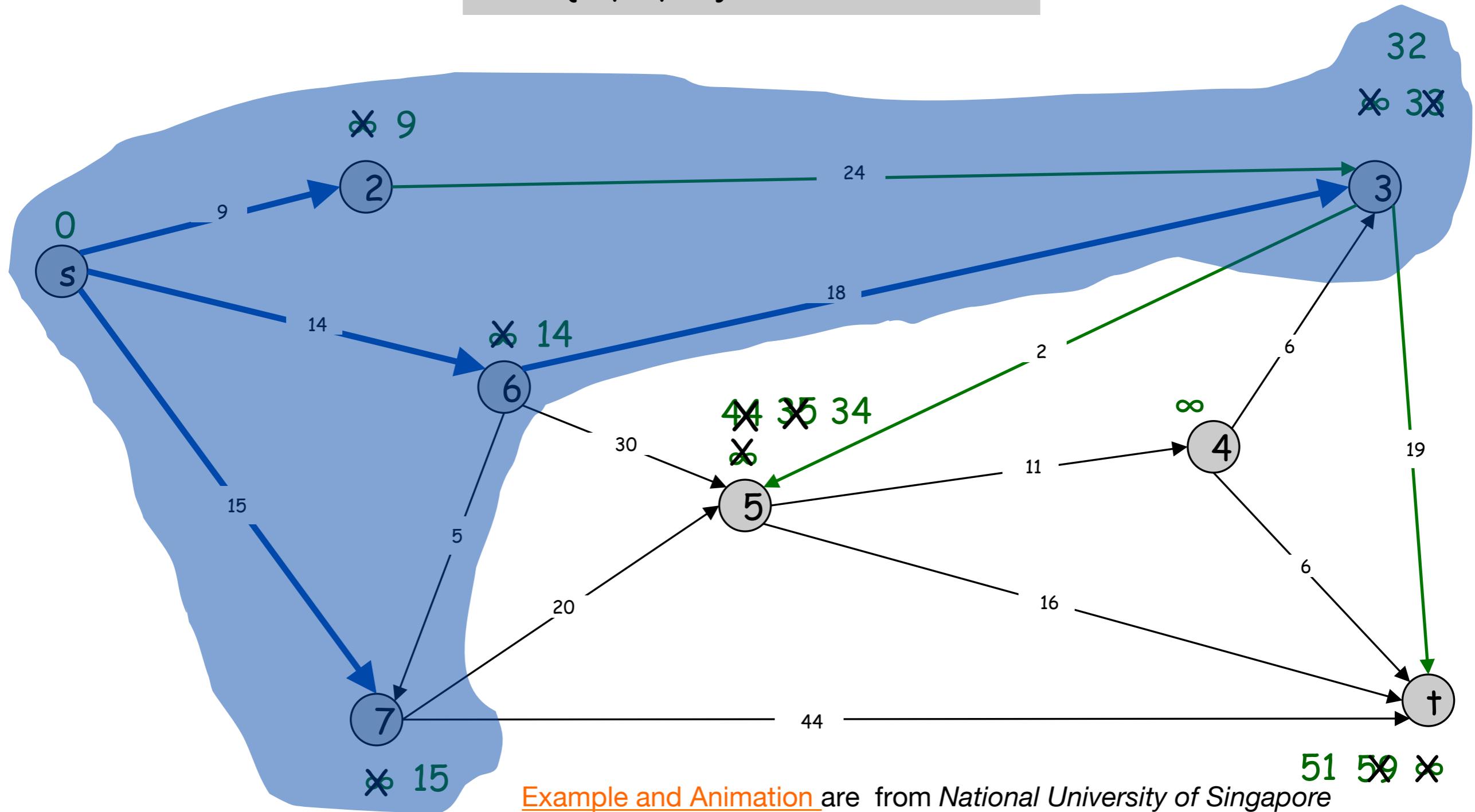
Dijkstra's Shortest Path Algorithm

$S = \{ s, 2, 6, 7 \}$
 $PQ = \{ 3, 4, 5, + \}$
 $B = \{ s, 7 \}$



Dijkstra's Shortest Path Algorithm

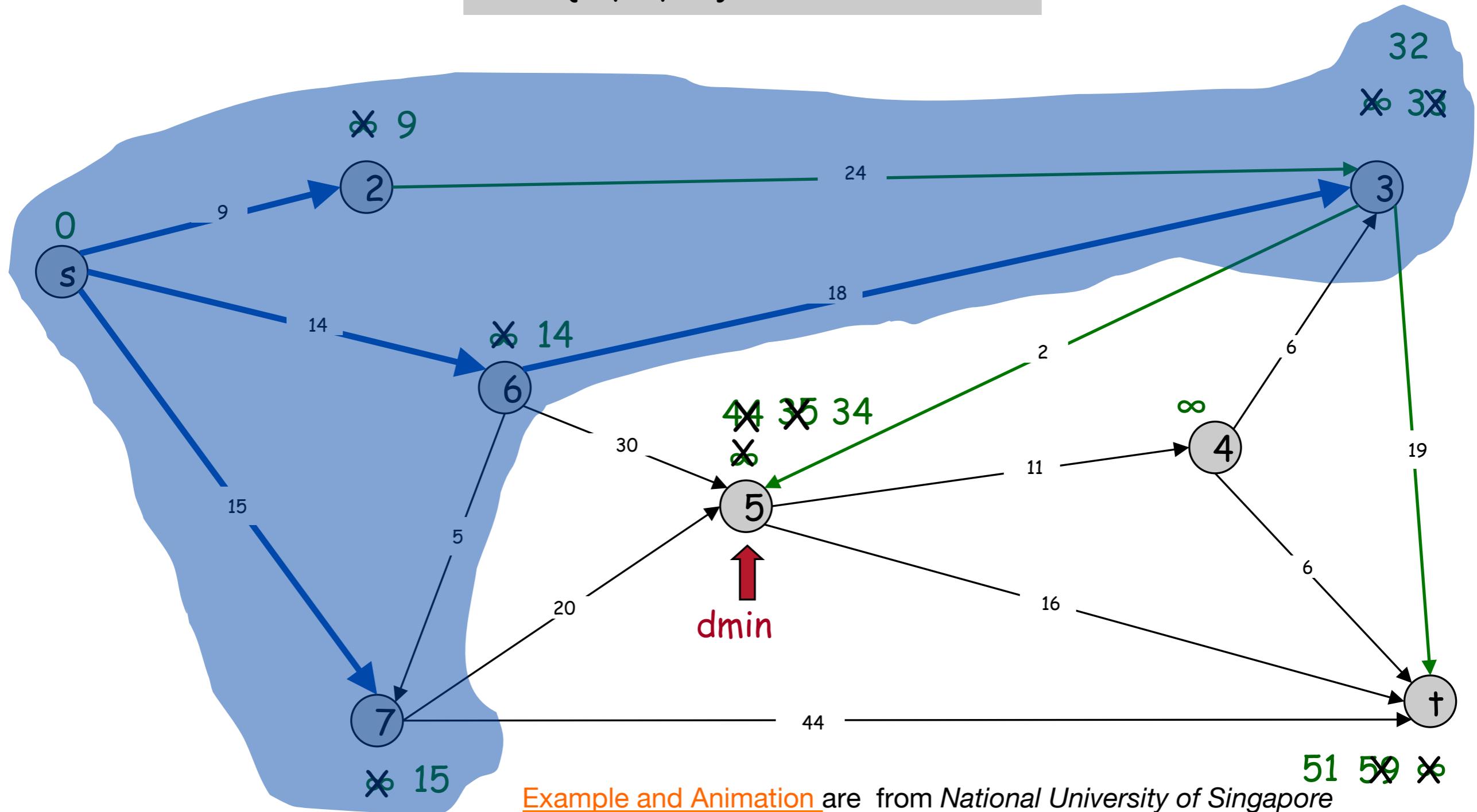
```
S = { s, 2, 3, 6, 7 }  
PQ = { 4, 5, + }  
B = { s, 6, 3 }
```



[Example and Animation](#) are from National University of Singapore

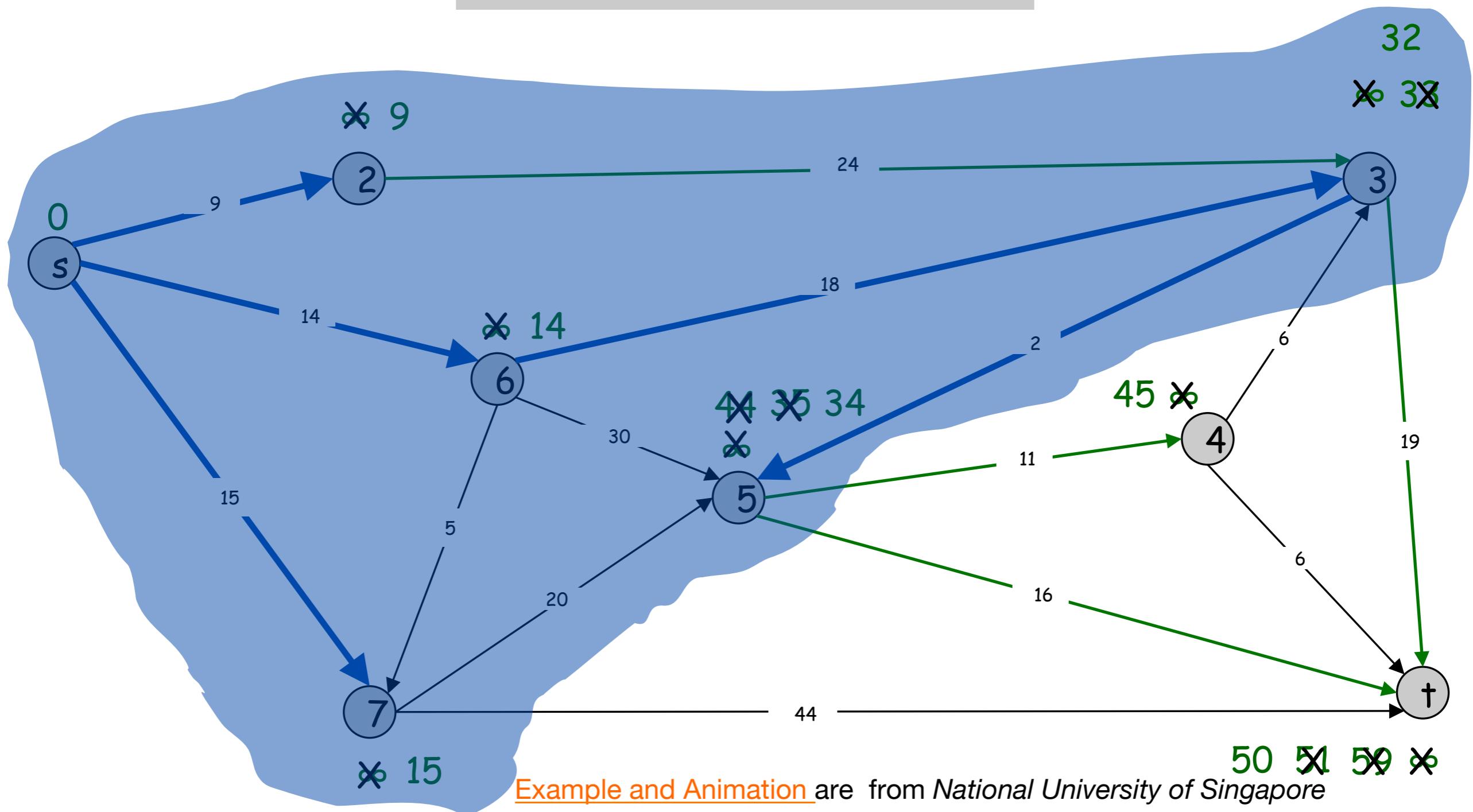
Dijkstra's Shortest Path Algorithm

```
S = { s, 2, 3, 6, 7 }  
PQ = { 4, 5, + }  
B = { s, 6, 3 }
```



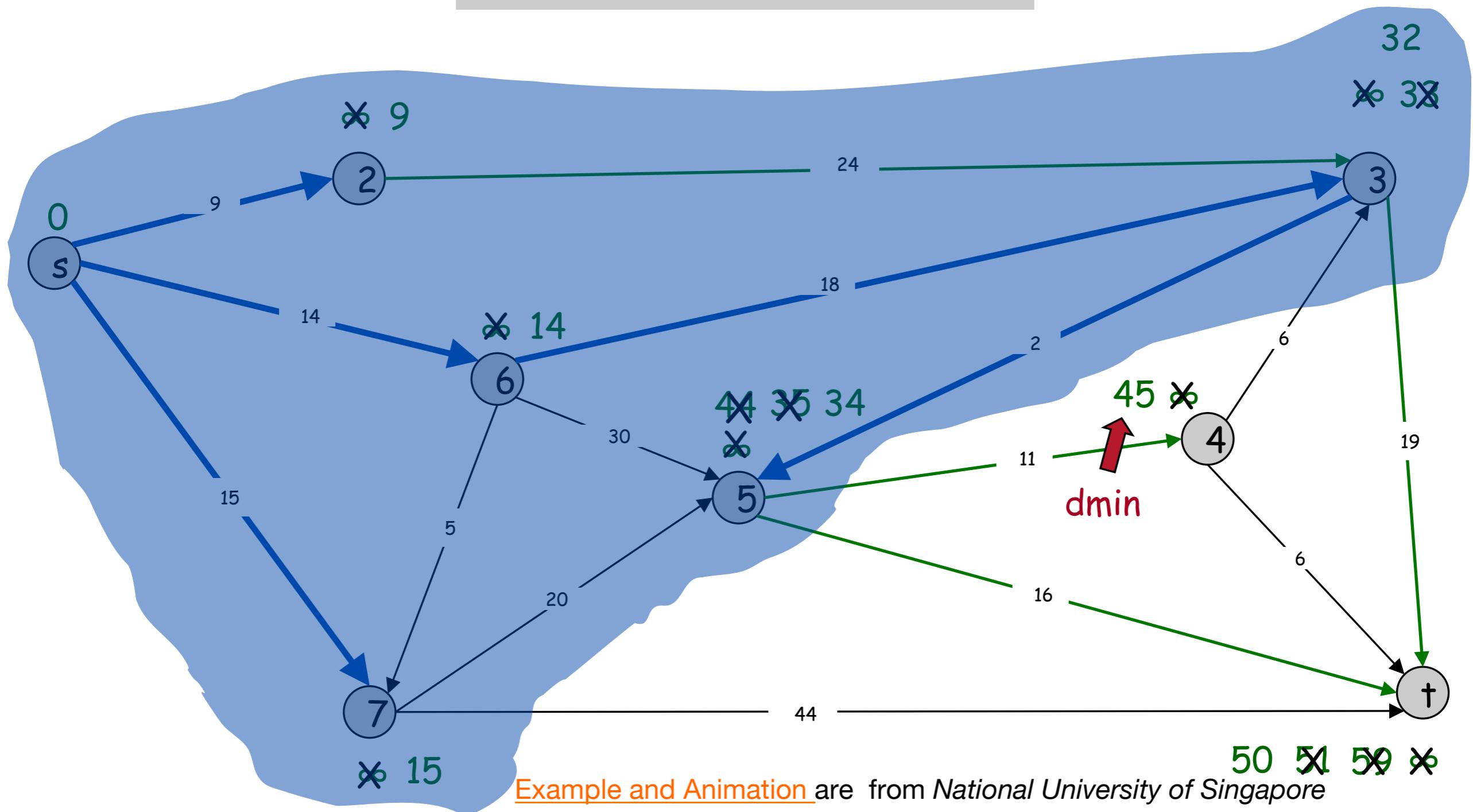
Dijkstra's Shortest Path Algorithm

```
S = { s, 2, 3, 5, 6, 7 }  
PQ = { 4, t }  
B = { s, 6, 3, 5 }
```



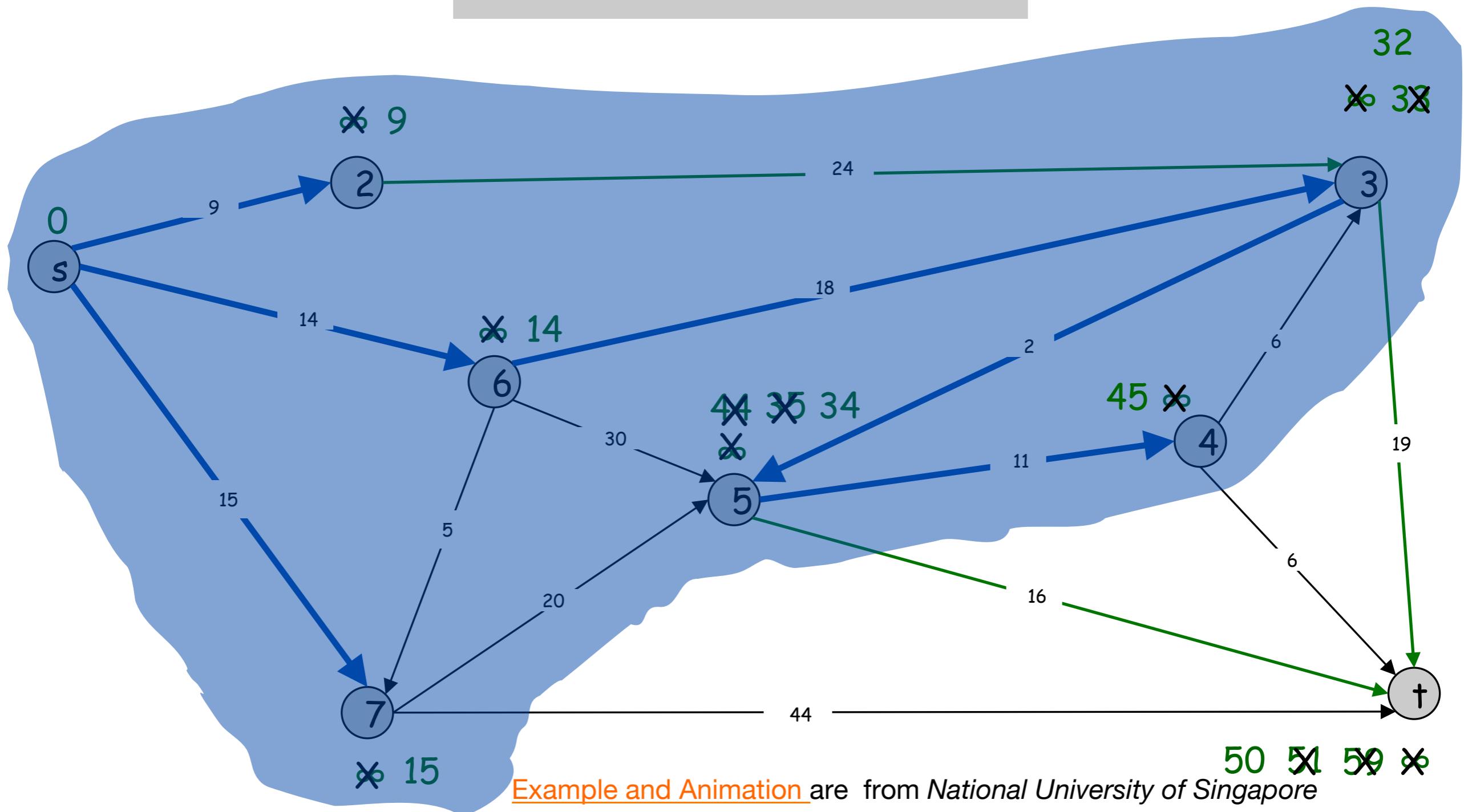
Dijkstra's Shortest Path Algorithm

```
S = { s, 2, 3, 5, 6, 7 }  
PQ = { 4, t }  
B = { s, 6, 3, 5 }
```



Dijkstra's Shortest Path Algorithm

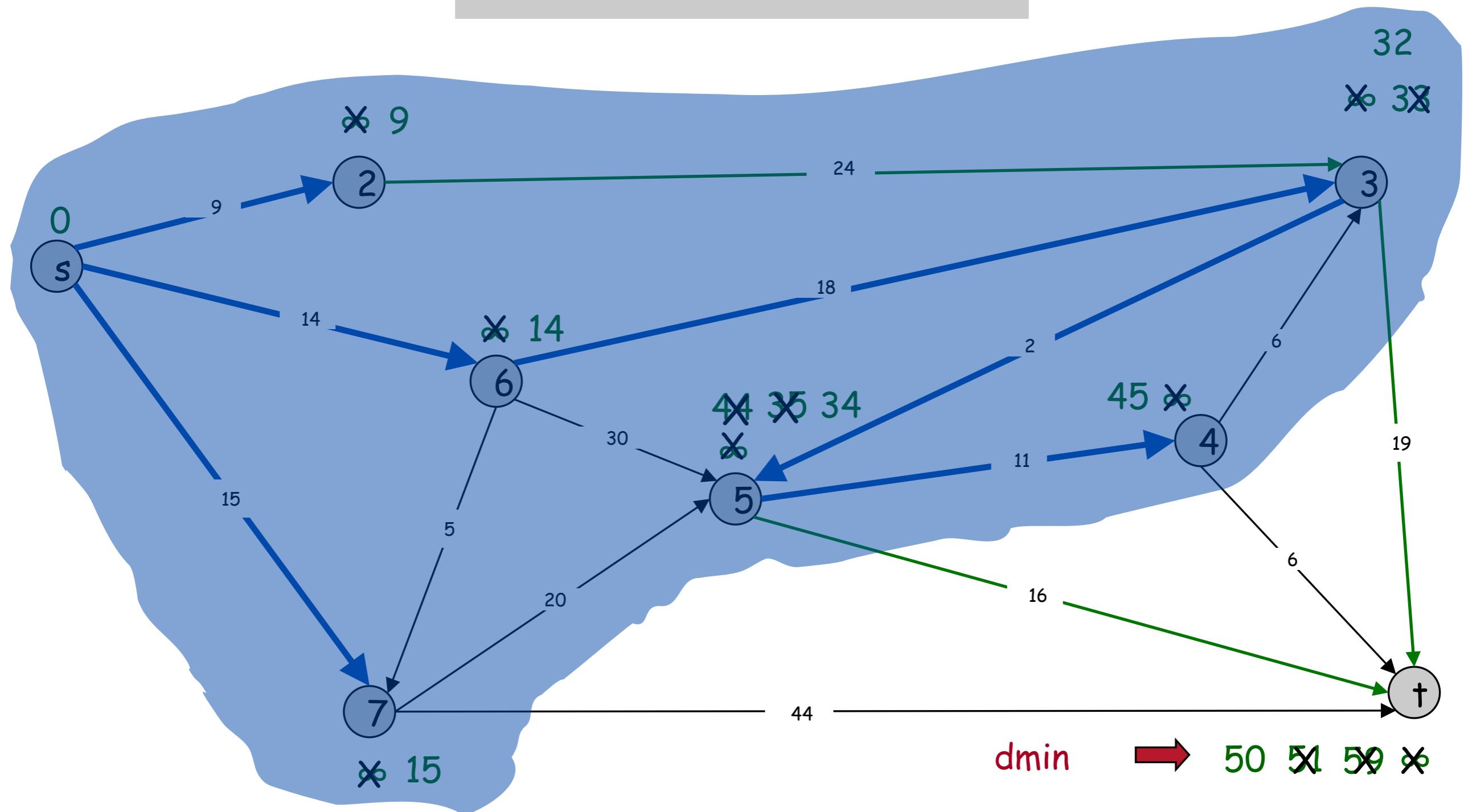
```
S = { s, 2, 3, 4, 5, 6, 7 }  
PQ = { + }  
B = { s, 6, 3, 5, 4 }
```



[Example and Animation](#) are from National University of Singapore

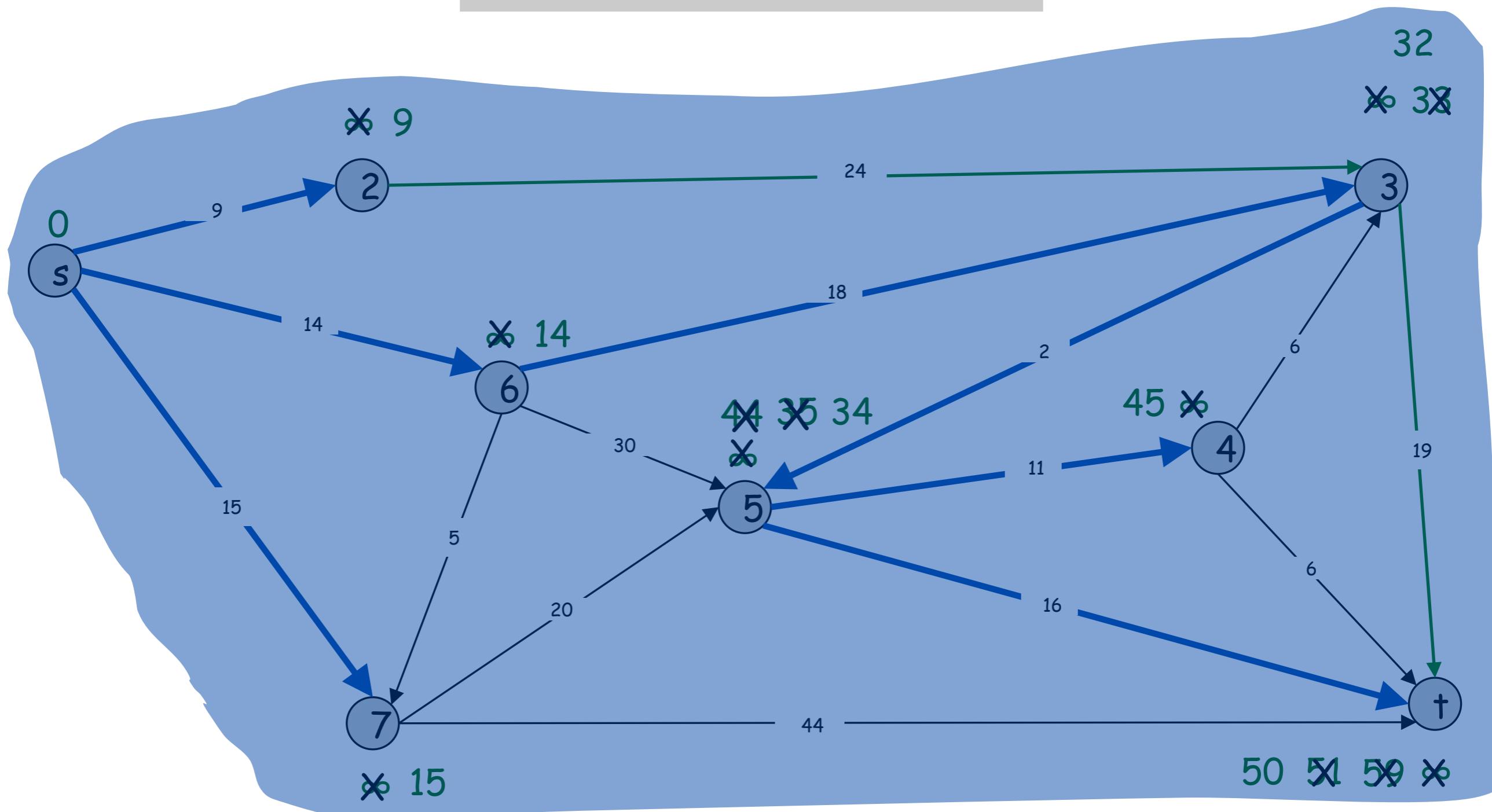
Dijkstra's Shortest Path Algorithm

```
S = { s, 2, 3, 4, 5, 6, 7 }  
PQ = { + }  
B = { s, 6, 3, 5, 4 }
```



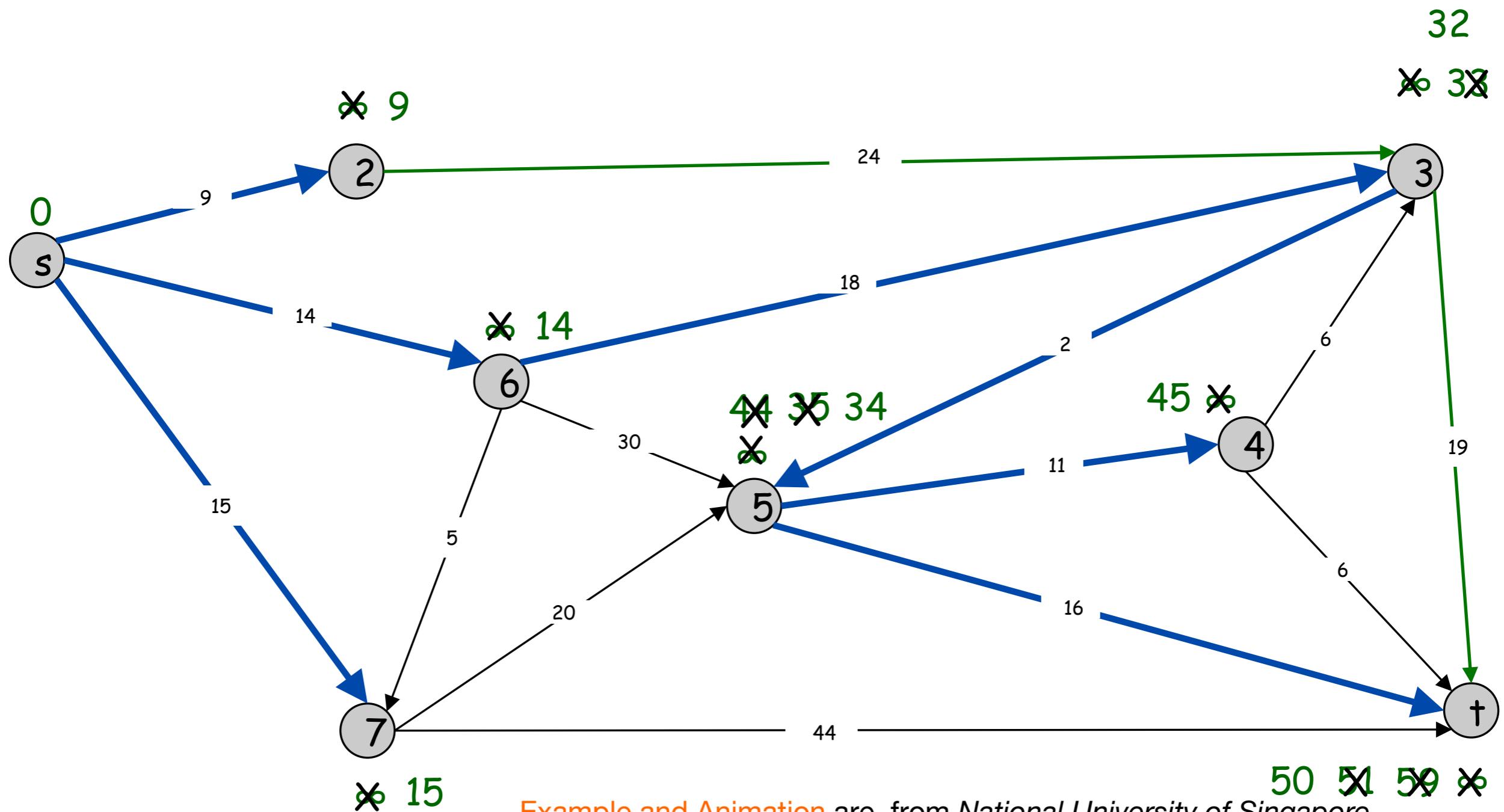
Dijkstra's Shortest Path Algorithm

```
S = { s, 2, 3, 4, 5, 6, 7, + }  
PQ = {}  
B = { s, 6, 3, 5, + }
```



Dijkstra's Shortest Path Algorithm

```
S = { s, 2, 3, 4, 5, 6, 7, † }  
PQ = { }  
B = { s, 6, 3, 5, † }
```



[Example and Animation](#) are from National University of Singapore

Dijkstra's Shortest Path Algorithm

```
function Dijkstra(Graph, source):
    dist[source] ← 0                                // Distance from source to source
    prev[source] ← undefined                         // Previous node in optimal path initialization

    for each vertex v in Graph: // Initialization
        if v ≠ source:                                // Where v has not yet been removed from Q (unvisited nodes)
            dist[v] ← infinity                      // Unknown distance function from source to v
            prev[v] ← undefined                     // Previous node in optimal path from source
        end if
        add v to Q                                    // All nodes initially in Q (unvisited nodes)
    end for

    while Q is not empty:
        u ← vertex in Q with min dist[u] // Source node in first case
        remove u from Q

        for each neighbor v of u:           // where v is still in Q.
            alt ← dist[u] + length(u, v)
            if alt < dist[v]:                // A shorter path to v has been found
                dist[v] ← alt
                prev[v] ← u
            end if
        end for
    end while

    return dist[], prev[]

end function
```

Importance of Efficiency

- Computers have finite speed, finite memory.
- Need to minimize amount of computation that the machines needs to make.

Algorithm X executes in $10n^2 + n / 3.7 + 22$ steps

Algorithm Y executes in $100n \log n + 14n + 22$ steps

Which algorithm is better?



Θ -notation

for $i = 0$ to n
 $a = a + 1$  $\Theta(n)$

for $i = 0$ to n
for $j = 0$ to n
 $a = a + 1$  $\Theta(n^2)$

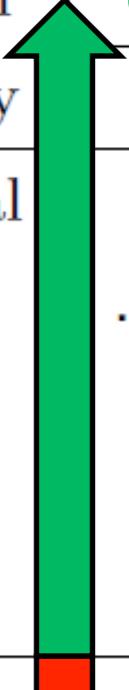
Algorithm X executes in $10n^2 + n / 3.7 + 22$ steps $\rightarrow \Theta(n^2)$

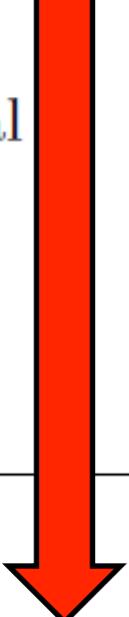
Algorithm Y executes in $100n \log n + 14n + 22$ steps $\rightarrow \Theta(n \log n)$

Which algorithm is better? **Algorithm Y**

Running Time Comparison

Algorithm complexity	Size n					
	10	20	30	40	50	60
polynomial						
n	.00001sec	.00002sec	.00003sec	.00004sec	.00005sec	.00006sec
n^2	.0001sec	.0004sec	.0009sec	.0016sec	.0025sec	.0036sec
n^3	.001sec	.008sec	.027sec	.064sec	.125sec	.216sec
n^5	.1sec	3.2sec	24.3sec	1.7min	5.2min	13min
super-polynomial						
2^n	.001sec	1.0sec	17.9min	12.7days	35.7yr	366cen
3^n	.059sec	58min	6.5yr	3855cen	$2 \cdot 10^8$ cen	$1 \cdot 10^{13}$ cen
$n!$	3.63sec	771cen	$8 \cdot 10^{16}$ cen	$3 \cdot 10^{32}$ cen	$1 \cdot 10^{49}$ cen	$3 \cdot 10^{66}$ cen

Good 

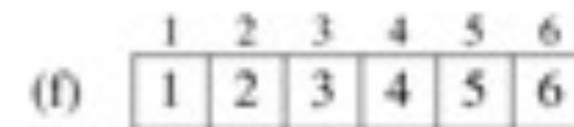
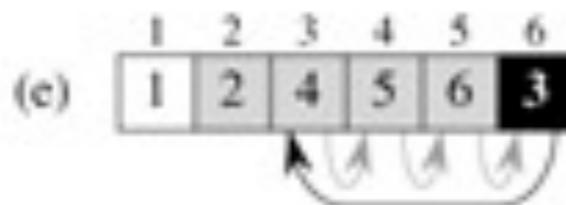
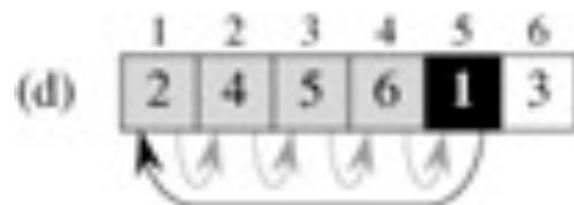
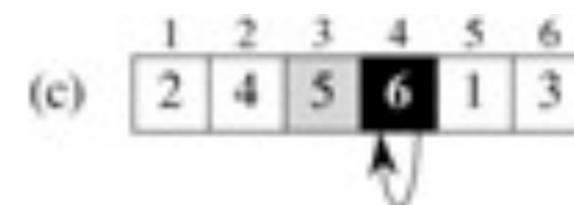
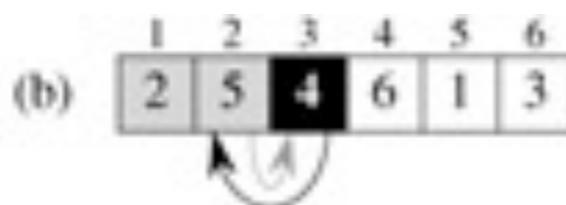
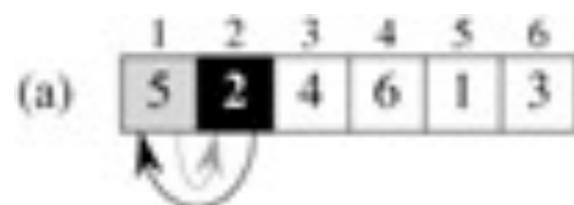
Bad! 

Sorting Algorithms

- Sorting is a fundamental operation in computer science and algorithms
- Many algorithms use it as an intermediate step
- There are several sorting algorithms, differing in their efficiency.

Insertion Sort

- Start with empty left hand
- Pick one card at a time
- Insert it into the correct position in left hand
 - To find correct position, compare it with each of the cards already in hand



Insertion Sort

```
INSERTION-SORT ( $A$ )
1  for  $j \leftarrow 2$  to  $\text{length}[A]$ 
2      do  $\text{key} \leftarrow A[j]$ 
3          ▷ Insert  $A[j]$  into the sorted sequence  $A[1 \square j - 1]$ .
4           $i \leftarrow j - 1$ 
5          while  $i > 0$  and  $A[i] > \text{key}$ 
6              do  $A[i + 1] \leftarrow A[i]$ 
7                   $i \leftarrow i - 1$ 
8           $A[i + 1] \leftarrow \text{key}$ 
```

Bubble Sort

- Go through the list:
 - If the next element is smaller:
 - SWAP!



unsorted



$5 > 1$, swap



$5 < 12$, ok



$12 > -5$, swap



$12 < 16$, ok



$1 < 5$, ok



$5 > -5$, swap



$5 < 12$, ok



$1 > -5$, swap



$1 < 5$, ok



$-5 < 1$, ok



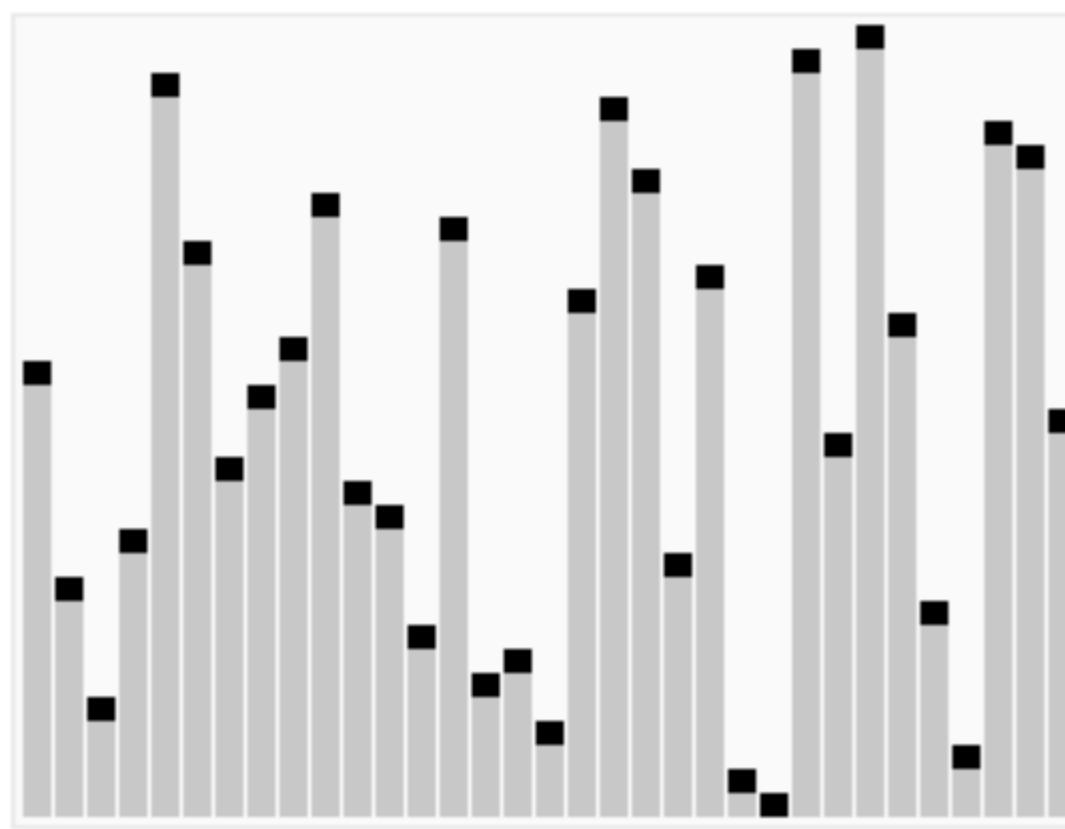
sorted

Bubble Sort

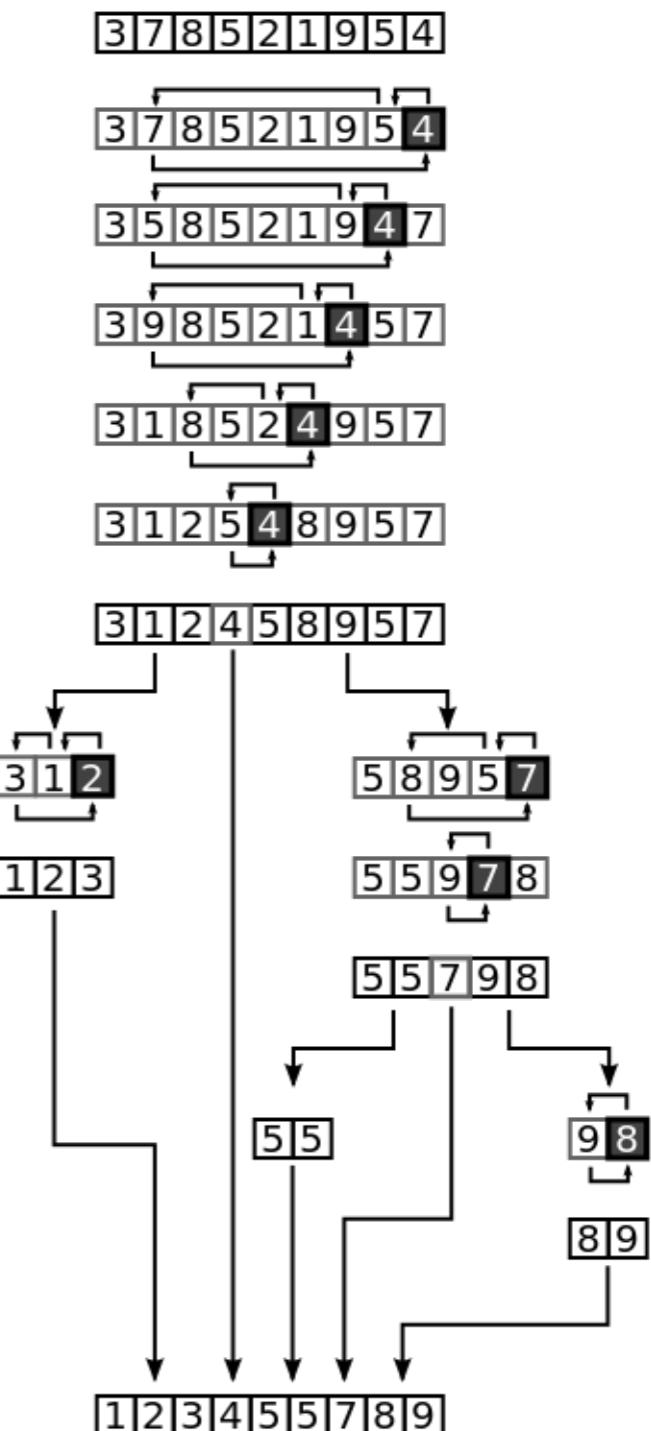
```
BUBBLESORT ( $A$ )
1 for  $i \leftarrow 1$  to  $\text{length}[A]$ 
2     do for  $j \leftarrow \text{length}[A]$  downto  $i + 1$ 
3         do if  $A[j] < A[j - 1]$ 
4             then exchange  $A[j] \leftrightarrow A[j - 1]$ 
```

Quick Sort

- Divide and conquer approach:
 - Pick an element from the list
 - Move elements smaller than the element before it
 - Move elements greater than it after it
 - Repeat for sub-problems



Animation and example from <https://en.wikipedia.org/wiki/Quicksort>
www.variability.org



Quicksort

```
QUICKSORT( $A, p, r$ )
```

```
1 if  $p < r$   
2   then  $q \leftarrow \text{PARTITION}(A, p, r)$   
3      $\text{QUICKSORT}(A, p, q - 1)$   
4      $\text{QUICKSORT}(A, q + 1, r)$ 
```

```
PARTITION( $A, p, r$ )  
1    $x \leftarrow A[r]$   
2    $i \leftarrow p - 1$   
3   for  $j \leftarrow p$  to  $r - 1$   
4     do if  $A[j] \leq x$   
5       then  $i \leftarrow i + 1$   
6         exchange  $A[i] \leftrightarrow A[j]$   
7   exchange  $A[i + 1] \leftrightarrow A[r]$   
8   return  $i + 1$ 
```



Exercise: Bubble sort and comparison to Python sorting algorithm

➤ Download material from

<https://ucla.box.com/s/c930rcapbp32ma0puuzkl5mt722zm18>

➤ We'll be using Jupyter Notebook for this exercise:

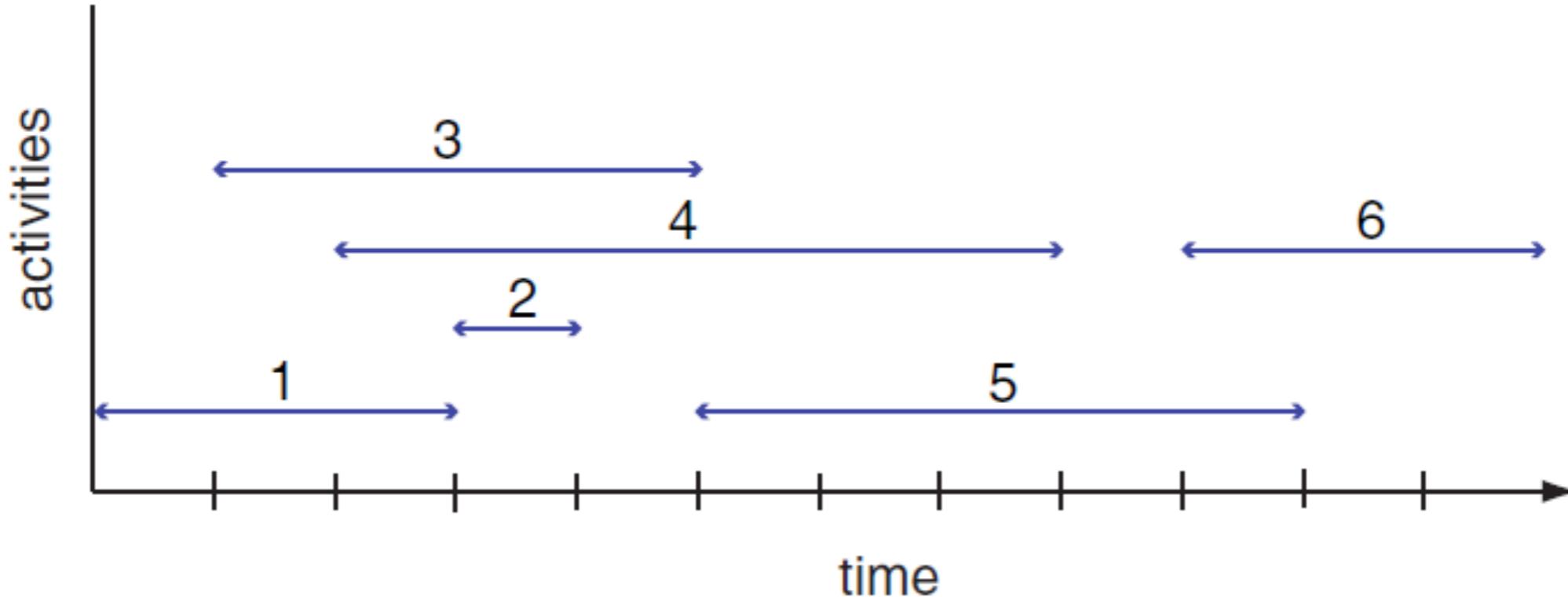
- Windows: Open “cmd”. OS X: Open terminal
- Type: cd <path to the sorting code>
- Type: jupyter notebook
- Open Sorting_Exercise.ipynb in the notebook
- Follow the instructions in the notebook
- functions.py: write your bubble sort function here



Algorithmic Techniques

- Brute-force algorithms
 - Take the most direct or obvious solution approach
 - Enumerate all possible candidate solutions
- Divide-and-conquer algorithms
 - break problem down into sub-problems that similar to the original problem, but smaller in size
 - Typically leads to recursive algorithms
 - E.g., Mergesort
- Greedy algorithms
 - Often applied to optimization problems involving a sequence of choices
 - At each step, the locally optimal choice is made
- Dynamic programming algorithms
 - Break problem down into sub-problems
 - Use the solutions to these sub-problems to solve larger sub-problems (reusing results)

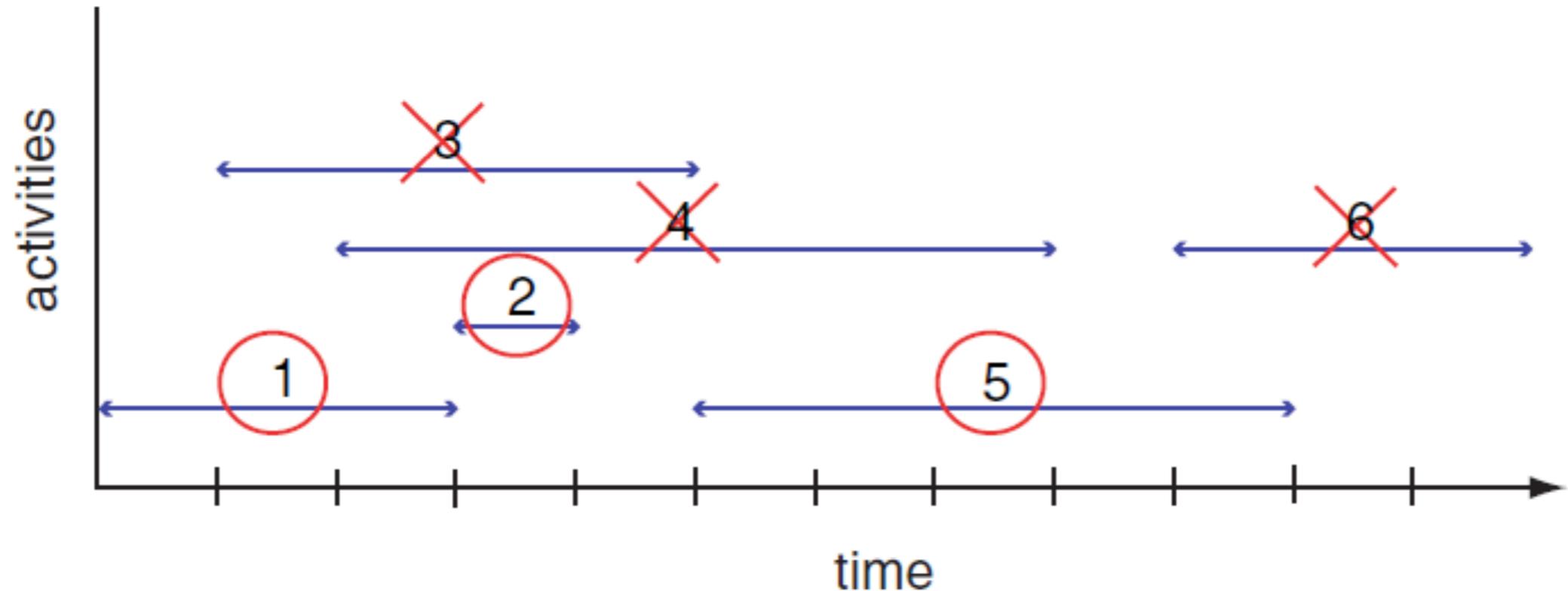
Greedy Algorithm



Activity Selection Problem

- **Given:** A set of proposed activities that wish to use a resource, which can only be used by one activity at a time
- **Problem:** Find maximum-size set of activities that do not have a time conflict

Greedy Algorithm



- **Greedy choice:** Pick the remaining compatible activity that has the earliest finish time
- Can you write an algorithm that does this?

Algorithm for Activity Selection Problem

ActivitySelector($s[n]$, $f[n]$)

```
1  $A \leftarrow \emptyset$ ;  $j \leftarrow 1$ 
2 for  $i \leftarrow 2$  to  $n$  do
3   if  $s_i \geq f_j$  then    ▷ greedy choice
4      $A \leftarrow A \cup \{i\}$ 
5      $j \leftarrow i$ 
6 return  $A$ 
```

- What is the running time for this algorithm?

Traveling Salesman Problem

- Given a list of cities and their pairwise distances,
 - find the shortest possible route that visits each city exactly once and returns to the origin city
 - How would you tackle this problem?



Mini-project: Obama's travel plan in Python

1. Think of a good algorithm to reduce the cost of Mr. Obama's trip

- Mr. Obama wants to visit 52 cities
- Travel cost= air fuel cost * distance
- Air fuel consumption rate= 5 gallons per mile (estimate for a Boeing 747)
- **Implement your algorithm in the python function provided**
 - Follow the instructions in Travelling_Salesman.ipynb
 - You are provided a function get_distance() to calculate distance between two locations.
- **Prepare 2-3 slides** describing your strategy and your final solution and total distance
- Can we solve it optimally?

2. Optional: Due to weather conditions, direct flights between some pairs of states are not possible.

- Find a generic solution
- As an example: these direct flights are blocked:
 - Austin, TX – Miami, FL
 - Washington D.C. – Baltimore, MD
 - Ann Arbor, MI – St Louis, MO
 - Vancouver, BC – Raleigh-NC
 - Atlanta, GA – Miami, FL

Mini-project: Obama's travel plan in Python

- Download code from

<https://ucla.box.com/s/a96y9cq0id741vnp7mponjiemeqfo3h5>

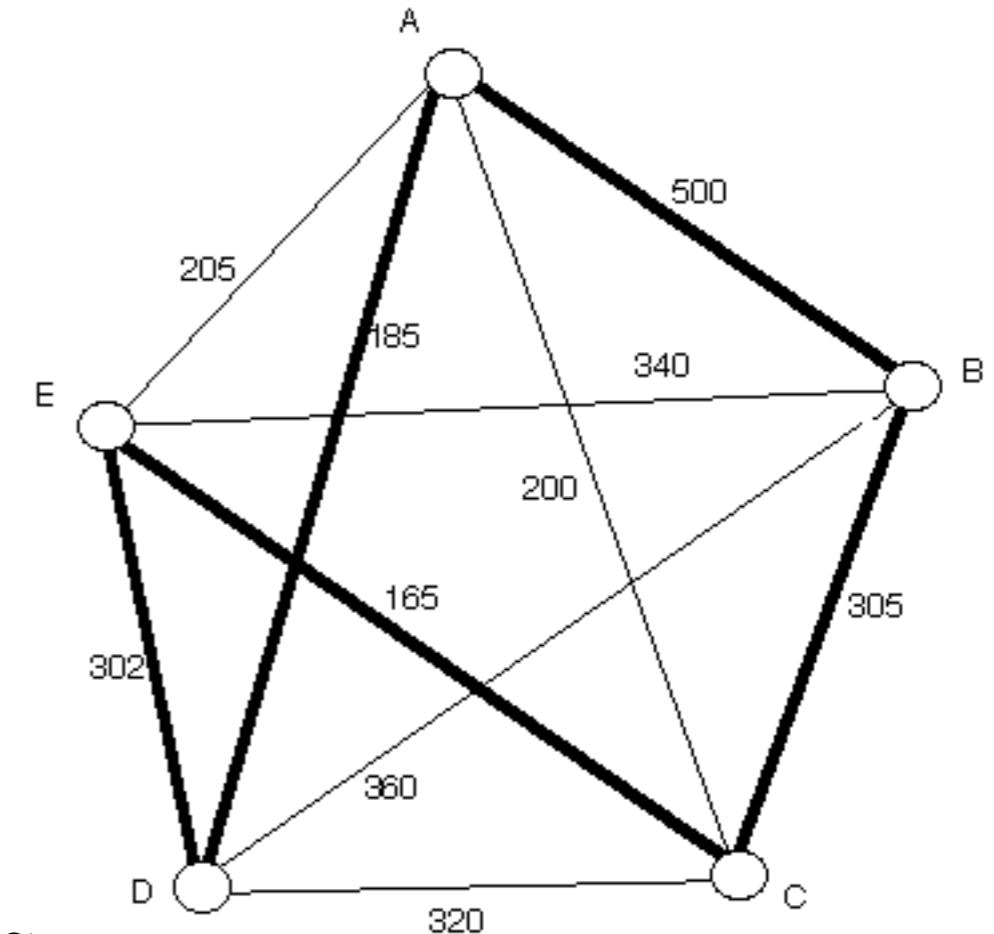
Submit code and slides by 3:00 pm on Thursday on Box.



Nearest Neighbor Algorithm

1. Start at a vertex
2. Find the lightest edge connecting the current vertex and an unvisited vertex V
3. Set current vertex to V
4. Mark V as visited
5. Terminate when all the vertices have been visited
6. Repeat from step 2

- Example: Starts from Vertex A and returns to starting point at the end
 - Path: A - D - E - C - B - A



Deficiencies of Nearest Neighbor

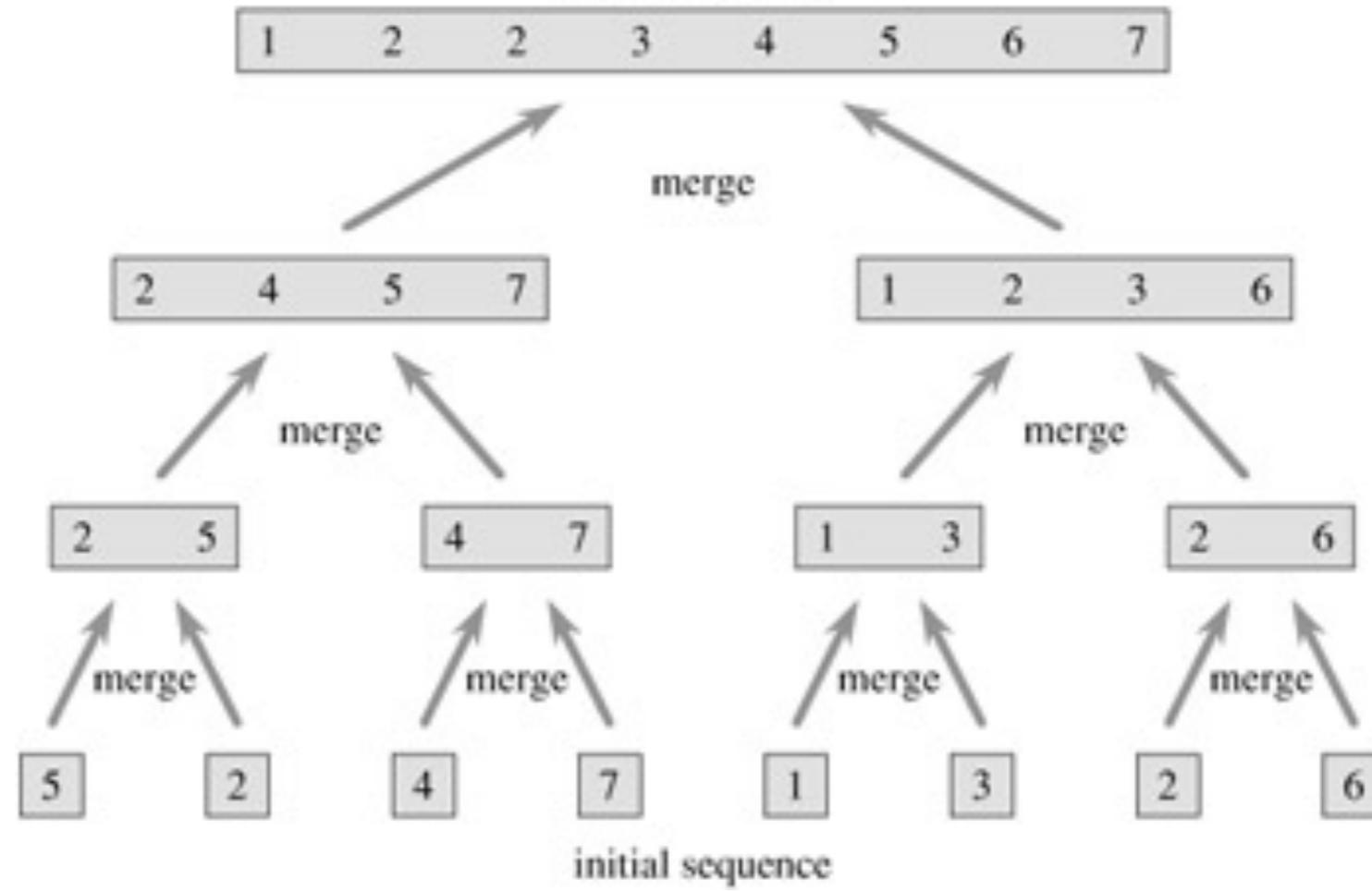
- Worst case: algorithm can result in a path much longer than the optimal path
- Won't work in case of incomplete graph
 - When some cities are not connected
- Any ideas for a better algorithm?



ADDITIONAL MATERIAL

Merge Sort

sorted sequence



- Divide and conquer algorithm
- Recursive algorithm
 - Split the list
 - Apply merge sort to the smaller list
 - After the smaller lists are collected: MERGE
 - (Note that each of the sublists is already sorted)

Mergesort

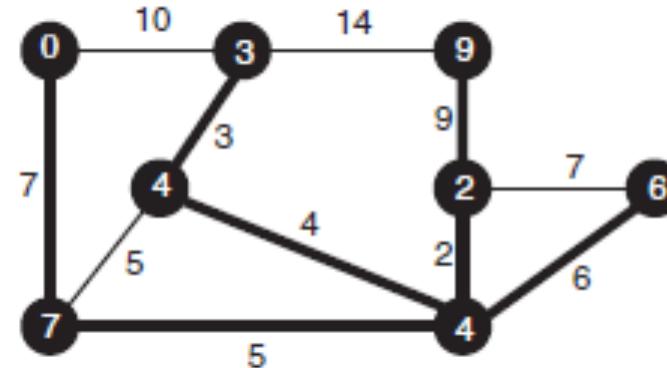
```
MERGE-SORT ( $A, p, r$ )
1 if  $p < r$ 
2   then  $q \leftarrow \lfloor (p + r) / 2 \rfloor$ 
3     MERGE-SORT ( $A, p, q$ )
4     MERGE-SORT ( $A, q + 1, r$ )
5     MERGE ( $A, p, q, r$ )
MERGE ( $A, p, q, r$ )
1    $n_1 \leftarrow q - p + 1$ 
2    $n_2 \leftarrow r - q$ 
3   create arrays  $L[1 \square n_1 + 1]$  and  $R[1 \square n_2 + 1]$ 
4   for  $i \leftarrow 1$  to  $n_1$ 
5     do  $L[i] \leftarrow A[p + i - 1]$ 
6   for  $j \leftarrow 1$  to  $n_2$ 
7     do  $R[j] \leftarrow A[q + j]$ 
8    $L[n_1 + 1] \leftarrow \infty$ 
9    $R[n_2 + 1] \leftarrow \infty$ 
10   $i \leftarrow 1$ 
11   $j \leftarrow 1$ 
12  for  $k \leftarrow p$  to  $r$ 
13    do if  $L[i] \leq R[j]$ 
14      then  $A[k] \leftarrow L[i]$ 
15       $i \leftarrow i + 1$ 
16    else  $A[k] \leftarrow R[j]$ 
17       $j \leftarrow j + 1$ 
```



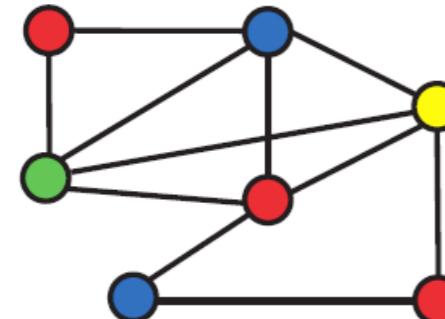
Optional Assignments

- Look into the following Algorithms

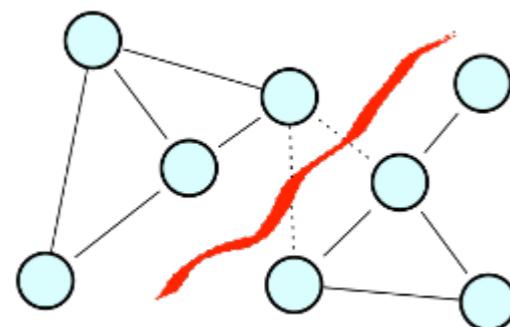
(1) Minimum spanning tree



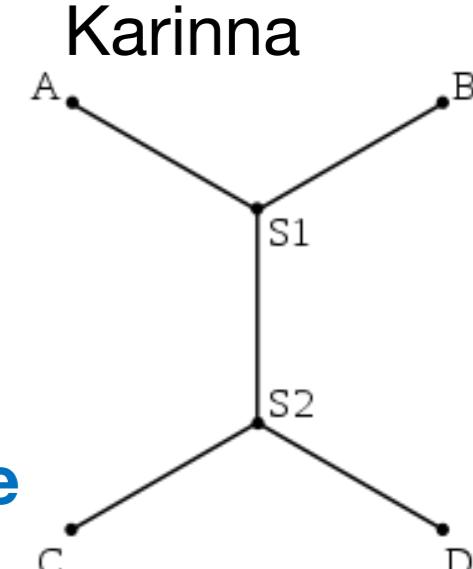
(3) Graph coloring problem



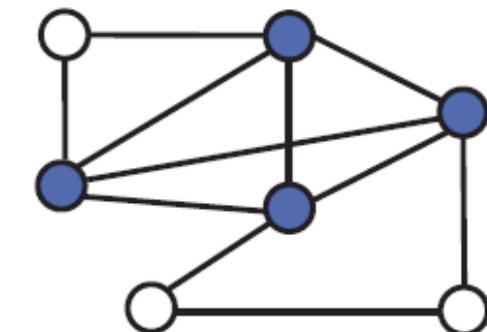
(4) Graph partitioning problem



(2) Minimum Steiner tree



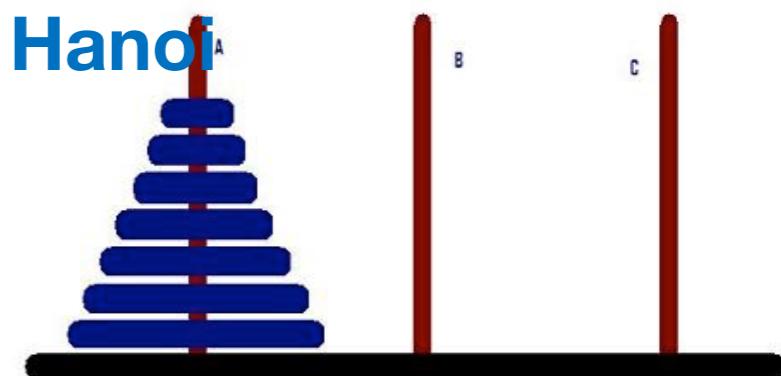
(4) Maximum Clique problem



(6) Knapsack problem



(7) Towers of Hanoi



Graph Exercises

- Graphs - nearest neighbor algorithm
 - <http://nodebox.net/code/index.php/Graph>

